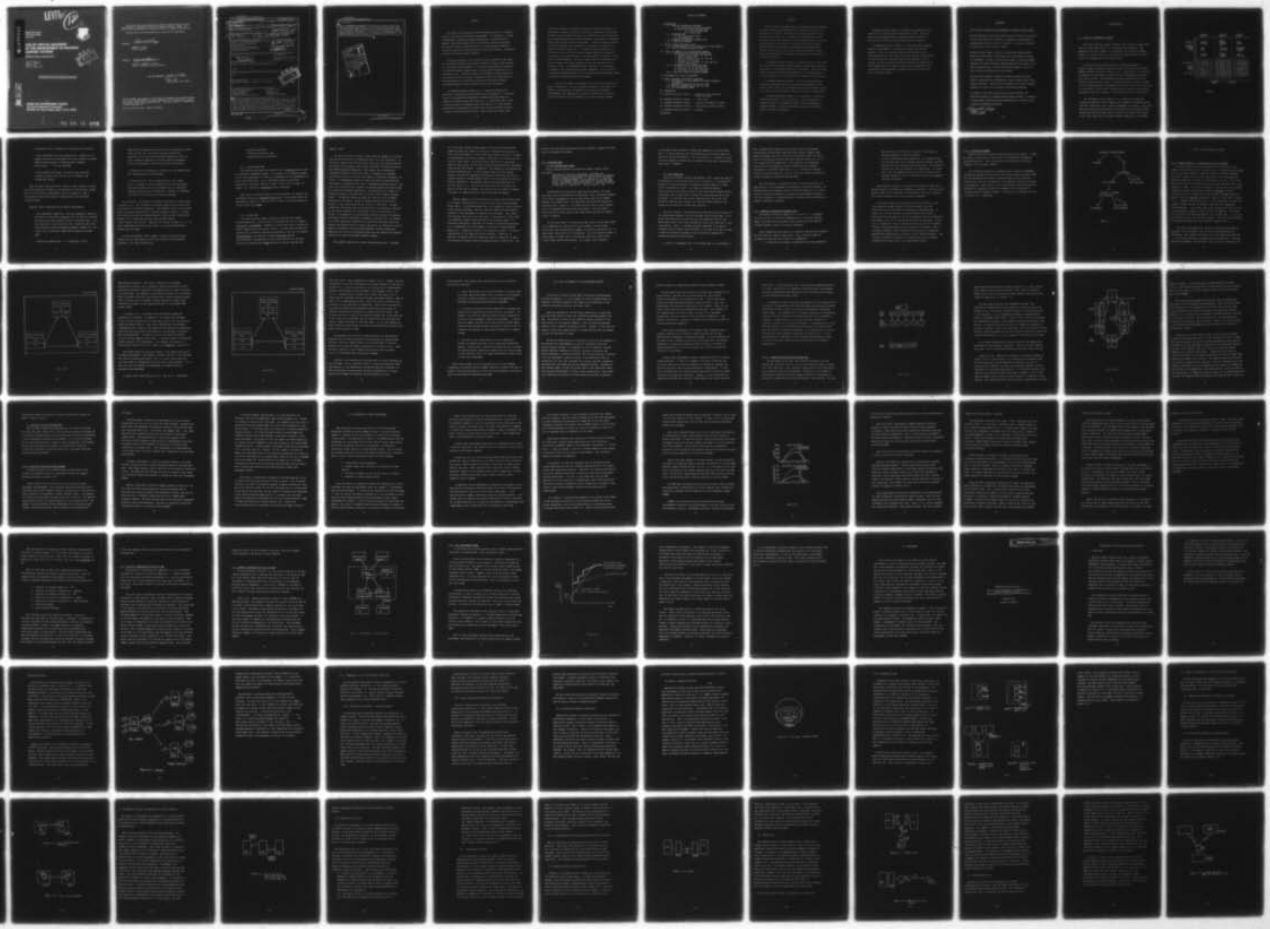
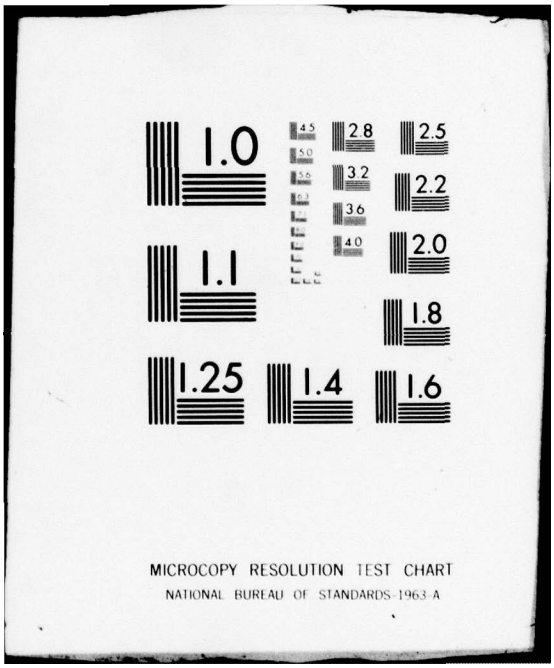
AD-A073 748 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
USE OF VIRTUAL MACHINES IN THE DEVELOPMENT OF DECISION SUPPORT --ETC(U)
JUL 79 J J DONOVAN, S E MADNICK, C LAM F30602-77-C-0205
UNCLASSIFIED RADC-TR-79-187 NL

1 OF 3
AD A073 748




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II

12

AD A073748

RADC-TR-79-187
Final Technical Report
July 1979

USE OF VIRTUAL MACHINES IN THE DEVELOPMENT OF DECISION SUPPORT SYSTEMS

MIT/Sloan School of Management

John J. Donovan
Stuart E. Madnick
Chat-Yu Lam
Tarek K. Abdel-Hamid

DDC
RECEIVED
SEP 17 1979
C

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

79 09 14 073

USE OF VIRTUAL MACHINES IN THE DEVELOPMENT

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-187 has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-79-187	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): USE OF VIRTUAL MACHINES IN THE DEVELOPMENT OF DECISION SUPPORT SYSTEMS	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. Sep 77 — Jan 79
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): John J. Donovan, Stuart E. Madnick, Chat-Yu Lam, Tarek K. Abdel-Hamid	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0205
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Center for Information Systems Research (CISR), MIT Sloan School of Management, Cambridge MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 55811006
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE), Griffiss AFB NY 13441	12. REPORT DATE: July 1979
	13. NUMBER OF PAGES: 251
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

Alfred P. Sloan

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

DDC RECEIVED SEP 17 1979 C

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Raymond A. Liuzzi (ISIE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Decision Support Systems	Operating Systems
Data Base Management	Computers
Virtual Machines	System Software
Management Information Systems	Programming Environments

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This effort is composed of several studies that have been conducted to investigate the development of a software programming environment for decision support systems using virtual machine technology as an integrating tool. The final final report combines the efforts of Reports 1-5 and highlights the merits of virtual machine technology as a software engineering tool. Report 1 examines various strategies for interfacing virtual machines. Report 2 examines these strategies and utilizes the CIS System to demonstrate their applicability. (Cont'd)

DD FORM 1 JAN 73 1473

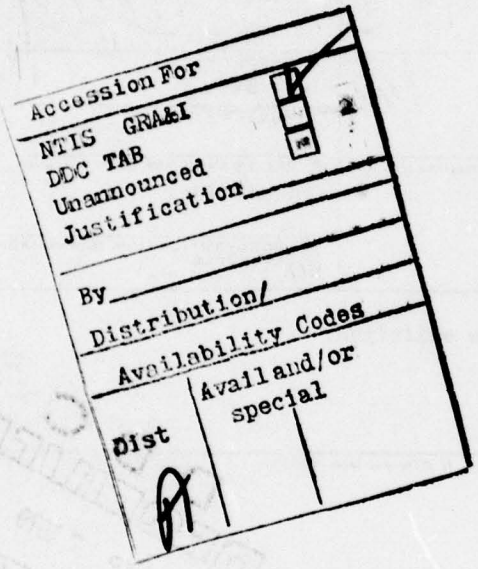
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409590

Item 20 (Cont'd)

Report 3 highlights the components of a Composite Information System (CIS). This system demonstrates multiple access to incompatible data bases using virtual machines. Report 4 utilizes the CIS system to demonstrate potential for enhancing decision-making in critical areas of SPO management. Finally, Report 5 develops the framework for a Composite Data System (COMPDATA). This system studies the interface problems of coordinating multiple-user access to multiple incompatible data base systems.



PREFACE

The Center for Information Systems Research (CISR) is a research center in the MIT Sloan School of Management. It consists of a group of Management Information Systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, generating and maintaining application software, information systems and decision support systems.

Within the context of the research effort sponsored by the Rome Air Development Center under contract F30602-77-C-0205, CISR proposed to investigate the development of a software programming environment for decision support systems using virtual machine technology as an integrating tool. Research issues to be studied included evaluating the potential utility of such a system for aiding Air Force SPO's in decision-making and formulating methods for providing automatic interfaces between virtual machines within this environment.

In Addendum Technical Report No. 1, various strategies for physically interfacing virtual machines are investigated. A detailed study of the Generalized Management Information System (GMIS) is reported in Addendum Technical Report No. 2. GMIS makes use of some of the strategies in Addendum Technical Report No. 1. that provide mechanisms for coordinating multiple user virtual machines accessing a common data base virtual machine. This study also reveals some

interesting semantic interfacing problems of GMIS-type systems. To gain further insight into both the physical and semantic interfacing problems (such as those that arise in coordinating access to multiple and incompatible data base virtual machines) the concept of Composite Information Systems (CIS) was developed. A study of several existing information systems that incorporate CIS concepts is reported in Addendum Technical Report No. 3. Addendum Technical Report No. 4 illustrates that a CIS-type decision support system has potential for enhancing the effectiveness of decision-making in certain critical areas of SPO management. Our brief study of SPO is intended to be used as a basis for future research to test the relevance of CIS concepts to SPO. In Addendum Technical Report No. 5, we develop a framework, the Composite Data System (COMPDATA), for the study of interfacing problems associated with coordinating multiply-user access to multiple incompatible data base systems.

In this final report, we attempt to put the entire study in its proper perspective. The many concepts discussed in the above five reports, and which are introduced in the context of different sub-problems, are combined into a general framework for the use of virtual machines in the development of decision support systems.

TABLE OF CONTENTS

I. INTRODUCTION
- I.1. TYPES OF INFORMATION SYSTEMS
  - A. Structured Decision Systems (SDS)
  - B. Decision Support Systems (DSS)
    - B.1. Institutional DSS
    - B.2. Ad-Hoc DSS
- I.2. DESIGNING DSS
  - A. The Conventional Tools
  - B. A New Approach
- I.3. COMPOSITE INFORMATION SYSTEMS (CIS)
- I.4. A CONCLUDING REMARK

II. USE OF VIRTUAL MACHINES IN DSS
- II.1. VIRTUAL MACHINES: AN INTRODUCTION TO THE CONCEPT
- II.2. USE OF VIRTUAL MACHINES IN DSS

III. GMIS: AN EXAMPLE OF AN EXPERIMENTAL SYSTEM
- III.1. COMMUNICATION MECHANISMS BETWEEN VMs
  - A. Communication between the user VM and the Manager VM
  - B. Communication between the user VM and the Interface VM
  - C. Communication between the Interface VM and the Data Base VM
- III.2. FUNCTIONS OF THE VIRTUAL MACHINES
  - A. Functions of the Manager VM
  - B. Functions of the Data Base VM
  - C. Functions of the Interface VM
- III.3. GMIS AS A DECISION SUPPORT SYSTEM

IV. AN EXAMPLE OF A GMIS APPLICATION

V. VIRTUAL MACHINES' INTERFACING PROBLEMS
- V.1. THE PHYSICAL vs THE SEMANTIC INTERFACE PROBLEM: AN ANALOGY
- V.2. PHYSICAL INTERFACING OF VMs IN A DSS
- V.3. SEMANTIC INTERFACING OF VMs IN A DSS
- V.4. VM's PERFORMANCE COSTS

VI. CONCLUSION

AI. ADDENDUM TECHNICAL REPORT I - STRATEGIES FOR INTERFACING VIRTUAL MACHINES

A2. ADDENDUM TECHNICAL REPORT 2 - A CASE STUDY

A3. ADDENDUM TECHNICAL REPORT 3 - COMPOSITE INFORMATION SYSTEMS

A4. ADDENDUM TECHNICAL REPORT 4 - SPO DECISION SUPPORT SYSTEM

A5. ADDENDUM TECHNICAL REPORT 5 - COMPDATA

REFERENCES

ABSTRACT

Decision Support Systems (DSS) are discussed with an emphasis on their technical requirements and the characteristics of the problems they address. The Composite Information System (CIS) approach is advocated as a framework for designing DSS, using virtual machine (VM) technology as an integrating tool. Relevant VM concepts are introduced, and examples of Decision Support Systems using virtual machine configurations are discussed. Special attention is given to the two major problems of using VMs in developing DSS, the physical interface and the semantic interface problems.

In addendum Technical Report 1, concepts useful to the study of virtual machine based Decision Support Systems are developed, strategies for interfacing machines are explored. These strategies focus on the port interfacing mechanism, i.e. getting output of one virtual machine to the input port of another virtual machine.

In Addendum Technical Report 2, several virtual machine interface mechanisms used in the Generalized Management Information System (GMIS) for breadboarding tools and database systems in an ad-hoc Decision Support System are studied in some detail.

In Addendum Technical Report 3, the Concept of a Composite Information System (CIS) is developed. The CIS approach is a general approach to integrate existing, often incompatible software system. Several case studies of CISs are presented under a consistent framework.

Addendum Technical Report 4 examines the organization and tasks of an Air Force System Program Office (SPO). Critical areas of SPO management where a Decision Support System can facilitate more effective decision-making are discussed.

In Addendum Technical Report 5, a high-level data language system called COMPDATA, is explored as a basis for further research. This data language system maintains application domain semantics so that a user is able to express a data request in terms of familiar concepts without any knowledge of data schema. The data language system also maintains information on the databases so that a user's request can be mapped onto the appropriate data manipulation operators on the data which may be in different systems.

EVALUATION

USE OF VIRTUAL MACHINES IN THE DEVELOPMENT OF DECISION SUPPORT SYSTEMS

The work described in the final technical report represents a significant reports (one through five) represents a significant accomplishment in accomplishment in establishing the framework of a software programming environment for decision support systems.

An ad-hoc decision support system is concerned with aiding decision-making for a wide variety of problems that are not usually recurring or anticipated. In order to develop such systems, it becomes extremely important to be able to integrate existing models and data bases, to be able to access the data in ways not previously thought of, and to be able to gather and assimilate new data quickly.

This effort has designed a Composite Information System (CIS) that permits integrating, often incompatible, software systems using virtual machine technology. This effort also examined how the decision support capability of the CIS can be applied to SPO decision problems.

The results of this effort have found that virtual machines are expecially suitable for building decision support systems. In addition, virtual machine technology has been demonstrated as a critical tool in a software programming environment.

Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

I. INTRODUCTION

I.1. TYPES OF INFORMATION SYSTEMS

The phrase "Decision Support System" (DSS) denotes a computerized system designed specifically to help people make decisions. The emphasis on the word "support" is important, since Decision Support Systems are usually viewed as tools which help people make decisions but do not automate decision-making per se.

To draw this distinction between the traditional use of computer-based information systems and uses in the field of Decision Support Systems, we refer to Figure (I.1.). Depicted in this figure is a framework for information systems that was developed by Gorry and Scott Morton (1971). This framework combines characterizations of Anthony (1965) and Simon (1960). Anthony's characterization is based on the proposed purpose of management activities (listed across the top of the matrix), while Simon's classification is based on the way that management deals with problems (listed along the side of the matrix).

The unshaded areas of Figure (I.1.) represent the types of information systems in which computers and computer technologies have been most effectively used to assist management in the past. Gorry and Scott Morton called these systems "Structured Decision Systems" (SDS) because the decisions these systems automate are largely structured. On the other hand, Decision Support Systems (depicted in the shaded

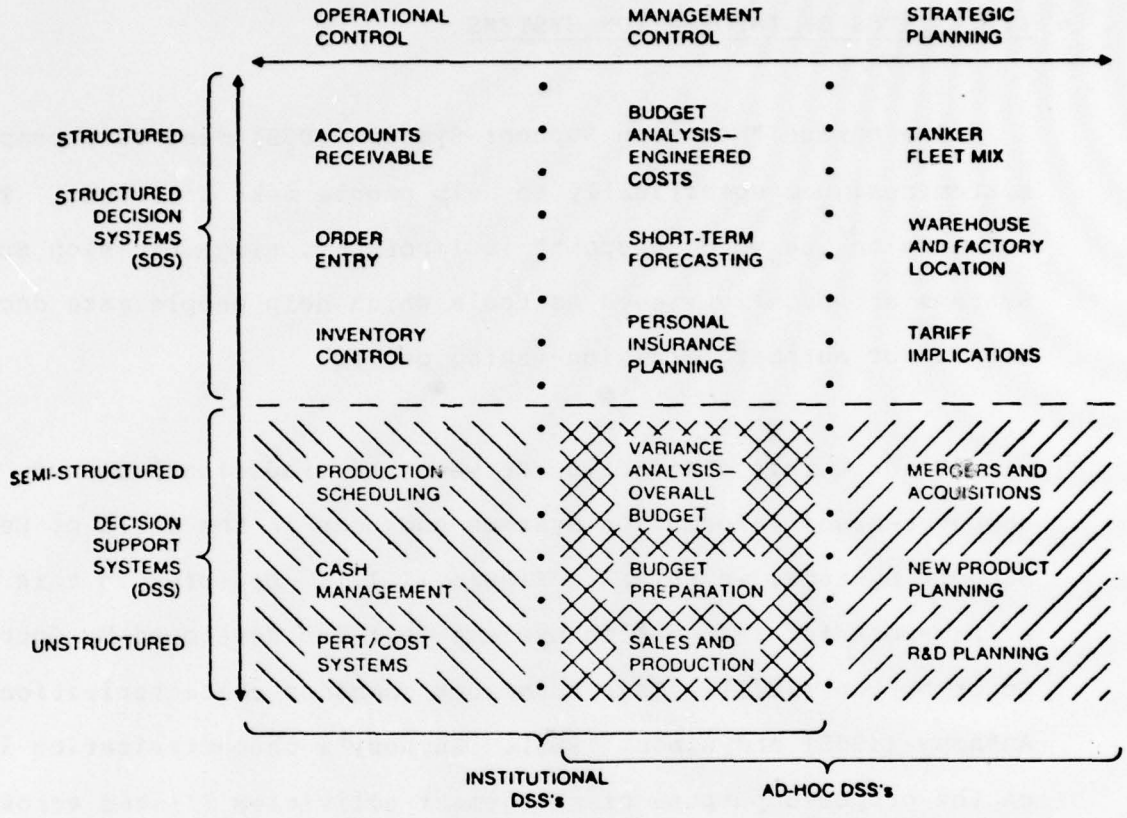

OPERATIONAL CONTROL
MANAGEMENT CONTROL
STRATEGIC PLANNING
STRUCTURED
STRUCTURED DECISION SYSTEMS (SDS)
SEMI-STRUCTURED
DECISION SUPPORT SYSTEMS (DSS)
UNSTRUCTURED
ACCOUNTS RECEIVABLE
ORDER ENTRY
INVENTORY CONTROL
BUDGET ANALYSIS—ENGINEERED COSTS
SHORT-TERM FORECASTING
PERSONAL INSURANCE PLANNING
TANKER FLEET MIX
WAREHOUSE AND FACTORY LOCATION
TARIFF IMPLICATIONS
PRODUCTION SCHEDULING
CASH MANAGEMENT
PERT/COST SYSTEMS
VARIANCE ANALYSIS—OVERALL BUDGET
BUDGET PREPARATION
SALES AND PRODUCTION
MERGERS AND ACQUISITIONS
NEW PRODUCT PLANNING
R&D PLANNING
INSTITUTIONAL DSS's
AD-HOC DSS's

Figure I.1

area) support largely unstructured decisions. These systems demand more than the traditional computational system can offer, and very little attention has been given to the technologies needed by such systems.

A. Structured Decision Systems (SDS):

A structured decision is one that is well-defined and is repeated often enough that decision rules or decision algorithms may be defined. The rules may be prespecified, and therefore they can generally be coded for computer processing. The use of computers to process structured decision rules represents a prior choice by a decision-maker as to how decisions are to be made in the future. Since decision-making is a costly process in terms of a very scarce resource --- management time and energy --- automating structured-type decisions is an efficient method for conserving the scarce resources and enhancing the productivity of management. The vast majority of the effort (and success) has been in developing computational systems that achieve this. In such a situation there is relatively little ambiguity, both to the system's designer and the system's user, as to the goals sought.

For example, the typical inventory control problem can be precisely stated, and it is clear what the criterion is by which solutions are to be judged. Other examples of SDS applications are shown in the upper half of the matrix of Figure (I.1.).

However, Gorry and Scott Morton (1971) argue that most of the

areas of greatest concern to managers, areas where decisions have a significant effect on the company, are in the lower half of the matrix. That is, managers deal for the most part with unstructered decisions.

B. Decision Support Systems (DSS):

If we were to extend the concluding remark of the previous section, we could add that computers and related systems which have so far been largely applied to the structured area have not yet had significant impact on management decision making. The areas of high potential do not lie in bigger and better systems of the kind most companies now use, but rather on information systems that are centered around the important decisions of the organization, many of which are unstructured.

But what are the characteristics of these unstructured problems which Decision Support Systems address? And what are the technical requirements of a DSS? In this section we shall attempt to answer these questions. We shall also present some examples of DSS applications.

With respect to the first question regarding the characteristics of unstructured problems, and based on the work of Scott Morton (1971) and of Donovan (1976), the following characteristics are emphasized:

- "Unprogrammed" nature of problems. There is no cut-and-dried method of handling the problem because it hasn't arisen before, or because its precise nature and structure are elusive or complex, or because it is

so important that it deserves a custom-tailored treatment.

- High requirements for data manipulation. Because of the complex nature of the problems, more than raw data is needed. Sophisticated analyses, transformations, displays, projections, ... etc, are required.

- Large complex data bases. The data could come from different sources, and therefore be in different and incompatible forms.

With the above characteristics in mind, we next attempt to answer the second question we posed at the beginning of this section, namely, what are the technical requirements that are needed in a DSS to "successfully" meet those types of problems that have the above characteristics?

Donovan (1976) recognized the following requirements:

- Data management capability. The data management capability must have an interactive component that can quickly introduce new data series and validate, protect, and query them. It is also important to have facilities for integrating data from diverse general purpose database management systems into such a form that a single-user application program may access them.

- Analytical capabilities. It is important to have

sophisticated computational facilities for analyzing the data since the data are continually changing and the the complexity of the problems addressed demand more than raw data. Required capabilities include modeling languages, statistical packages, and other analytical facilities.

- Reliability, maintainabilty, flexibility, and capabilities for incorporating new technologies.

- Transferability. This is important for two reasons. First, it allows for the incorporation of existing models, data base systems, and other software into an integrated framework (and quickly if necessary). Second, it minimizes the need to re-train users of existing systems.

At this point, the reader is seriously encouraged to reflect back to Figure (I.1.) and re-examine the example applications given within the context of the above discussion. The examples reveal the very wide range of prospect applications of Decision Support Systems. To allow for a better analysis and understanding of these systems, Donovan and Madnick (1977) conceived two major types of DSS, namely, "institutional" and "ad-hoc." However, by looking at Figure (I.1.), we would immediately anticipate that there is no precise distinction between the two types.

Donovan and Madnick (1977) suggest a means of differentiating between the two types based on three key features of the problems addressed. The three features are:

- Problem recurrence,
- Problem anticipation, and
- Problem definition stability.

B.1. Institutional DSS

Institutional DSS deal with decisions of a recurring nature, such as a portfolio management system. For example, an investment manager in the trust department of a bank is responsible for the management of one or more portfolios. He is repeatedly faced with decision situations of the type, "Should I trade in stock "X" for stock "Y" today?" He, therefore, anticipates these situations.

With such a continuous contact with the problem, the manager will undoubtedly develop a reasonable definition of his problem and of the approaches to solve it. And once this is done, the problem defintion remains relatively stable.

B.2. Ad-Hoc DSS

An ad-hoc decision support system is concerned with aiding decision-making for a wide variety of problems that are not usually recurring or anticipated. Examples in the private sector include new product opportunities and merger offers. In such cases the specific problem and its decision type being addressed at any time are usually poorly defined, the decision is needed very soon and the decision-makers' perceptions of the problem and even the inherent nature of the problem may change during the process (Donovan and

Madnick, 1977).

We would like now to present a more detailed example of an ad-hoc type DSS and which we will repeatedly refer to in our discussion. It involves a request that the MIT Energy Laboratory and CISR received at the height of the energy crisis during the winter of 1973-74 from the New England Regional Commission. The request was to develop an information system to assist the region in managing the possible distribution of oil to minimize the impact of shortages throughout the region. A considerable amount of effort was spent in designing and developing a prototype of just such a shortage information system. But less than six months later, before the system became fully operational, the problem had changed completeley. New England was no longer in a shortage situation, as there was a backlog of full tankers in Boston Harbor. Instead the region was beset by a new series of problems, namely price increases. Prices of energy had gone up by over 50 percent in that three-month period (winter of 1973-1974). Certain industries and sectors within the region were thus adversely affected. As the region realized its vulnaribilty to price fluctuations in energy, the problems of the policymaker shifted from ones of handling shortages to ones of analyzing: methods to conserve fuel, the impacts of tariffs, decontrol, effects of oil prices on different industrial sectors and states within the region, the merits of refineries, and the impacts of offshore drilling on New England's fishing industries. These are but a few of the problem areas which New England policymakers faced and on which they needed immediate support.

This example typifies the ad-hoc DSS characteristics. Although

all of the work centers around energy, the actual decisions being supported range from issues of taxation to conservation policies in state office buildings, to tariff and import-export controls, to impact on unemployement and general economic conditions. As a result of the range of these topics, the precise data required for a particular decision type as well as the kind of analysis and analytical techniques required can vary widely. In addition, the answers to the problems are usually needed quickly. It was necessary that the state energy officers, as well as legislators and governor's officers, be able to respond rapidly to energy problems and to initiate regional as well as federal legislation. Furthermore, most of the decisions were of a one-time nature, i.e. once the particular decision was made, the specific ad-hoc DSS to support that decision may no longer be needed (although its components may be used for related decisions).

Another example of a decision-making situation that exhibits most of the above characteristics is that of an Air Force System Program Office (SPO). In Technical-Report No. 4 we studied the characteristics of SPO decision problems, and the resources (models and data bases) available to SPO management to address those problems. Here, again, the decisions that have to be made cover a wide spectrum. They range from handling a budget reduction made by the funding agency; to evaluating contractors' proposals for changes in engineering designs; to a re-scheduling of certain phases in the face of unanticipated delays (e.g. caused by a strike in a contractor's plant); to even a redefinition of the scope of the entire project. There is, in addition, a constant need for answering "what-if" questions in short time notices for project monitoring and/or project re-direction. And

finally, the data bases and analytical tools needed to support the wide range of problems vary widely.

I.2. DESIGNING DSS:

A. The conventional Tools:

After a detailed study of 56 decision support systems, Alter concluded:

> "Although the (current) technology is adequate for applications involving single data bases, simple time-shared models, standard calculations ... and so on, it is quite lacking as regards access to and manipulation of data from large or broad data bases, development of representational models, development and modifications of large-scale systems of any type ... and so on." (Alter, 1975, p. 18)

Considering first the ad-hoc type decision systems, one would seriously doubt the feasibility of building a total information system that contains initially all of the facilities likely to be needed. Besides the enormous cost, time and effort that would be required to construct such a system, it is highly unlikely that it could contain all of the needed facilities (data and models), since the basic definition of an ad-hoc DSS is that these facilities are not usually known in advance.

On the other hand, in institutional DSS, where the decision types to be supported are recurring and fairly stable, the particular facilities that can effectively support the decision making can be, in many cases, quite difficult to determine. The decision-makers need, in many cases, to "play" with the system for some time, before their "real" needs could be determined. As a result, after spending

considerable time and money to design and implement an institutional DSS, it is still possible that the system may not be found useful or usable by the decision-makers. For this reason, it is typical to find that many changes and additions are needed to bring such a system from operational to usable.

B. A New Approach:

Donovan and Madnick (Donovan and Madnick, 1977) argue that many of the problems of institutional DSS can be relieved if it is possible to develop a flexible prototype or breadboard system rapidly, in weeks rather than months or years. Its impact on decision-making effectiveness can be better evaluated, and operational problems can be identified, and in many cases, resolved by rapid modification of the prototype. In this way, the actual implementation costs and risks should be significantly reduced, and expected effectiveness of the system can be much more reliable and realistic.

The tools needed to help develop an ad-hoc DSS are very similar to the requirements needed to build a prototype institutional DSS. In an ad-hoc DSS time and low fixed costs are the important factors, not operational costs, since these systems are often only used once or very infrequently. Hence, it becomes extremely important to be able to integrate existing models and data bases, to be able to access the data in ways not previously thought of and to be able to gather and assimilate new data series quickly.

In order to accomplish this, it was found that it is necessary to

have a computational capability that allows rapid and unplanned assimiliation of data bases (and data base systems), models (and modeling systems) and other facilities that may be useful in a DSS - even if such systems had never been used together before and are, in fact, operationally incompatible. This approach of integrating existing, often incompatble software systems has been studied extensively in Technical Report No. 3. It is called the Composite Information Systems approach, and the resulting information system is called a Composite Information System (CIS).

The CIS approach, it should be noted, is not restricted in its applicabilty to the development of management information systems. As demonstrated by the examples cited in Technical Report No. 3 it has a rather wide range of applications ranging from developing systems to aid civil engineers in problem-solving to systems used by economists to conduct econometric analyses.

I.3. COMPOSITE INFORMATION SYSTEMS (CIS):

As defined in Technical Report No. 3 a CIS is " ...a computer facility to host a variety of components (e.g. data base systems, models, and application programs) that may have been developed by different people, often for different purposes."

These components often operate in different operating environments and use different types of data bases. Thus, two basic types of incompatibilities may exist between the components:

(1) Operating environment incompatibilities among components

may arise due to differences in execution environments, operating systems, or the host machine.

(2) Data differences among components. Two components may use the same type of file system, operating system, and host machine but each encodes the data differently, thus making it difficult for the two components to communicate. Or, worse still, the two components may use entirely different file systems, making the task of integrating these two components even more difficult.

These problems, however, are solvable and Technical Report No. 3 presents several strategies for component integration. The rest of the present report will be devoted to a discussion of one such strategy, that which is based on the virtual machine technique.

In the next chapter, we shall present an introduction to the virual machine concept together with some insights into the effectiveness of virtual machine technology in designing DSS. In chapter III, we then present a detailed examination of GMIS, a particular DSS that makes use of the CIS approach and VM technologies. Our emphasis here will be the design features of the system. We will then, in chapter IV, look at the operational features of the system through a detailed presentation of an application example. This is followed by chapter V which examines the more general problems of employing VM technologies to construct CIS for decision support. And finally we conclude in chapter VI by indicating the key research problems that we feel have to be investigated further.

I.4. A CONCLUDING REMARK:

Before going into the more technical parts of our report, it might be useful at this point to take stock of the different concepts we presented thus far. This, we believe, will help place the coming chapters in their proper context.

We started by presenting a framework that classifies management information systems into Structured and Decision Support Systems (DSS). We then addressed the problem of designing DSS, and for which we advocated the use of a new approach, the Composite Information Systems (CIS) approach. And, finally, the virtual machine (VM) technique was proposed as a possible strategy to implement the CIS approach in designing Decision Support Systems. Pictorially, we followed the tree of Figure (I.2.) downwards.

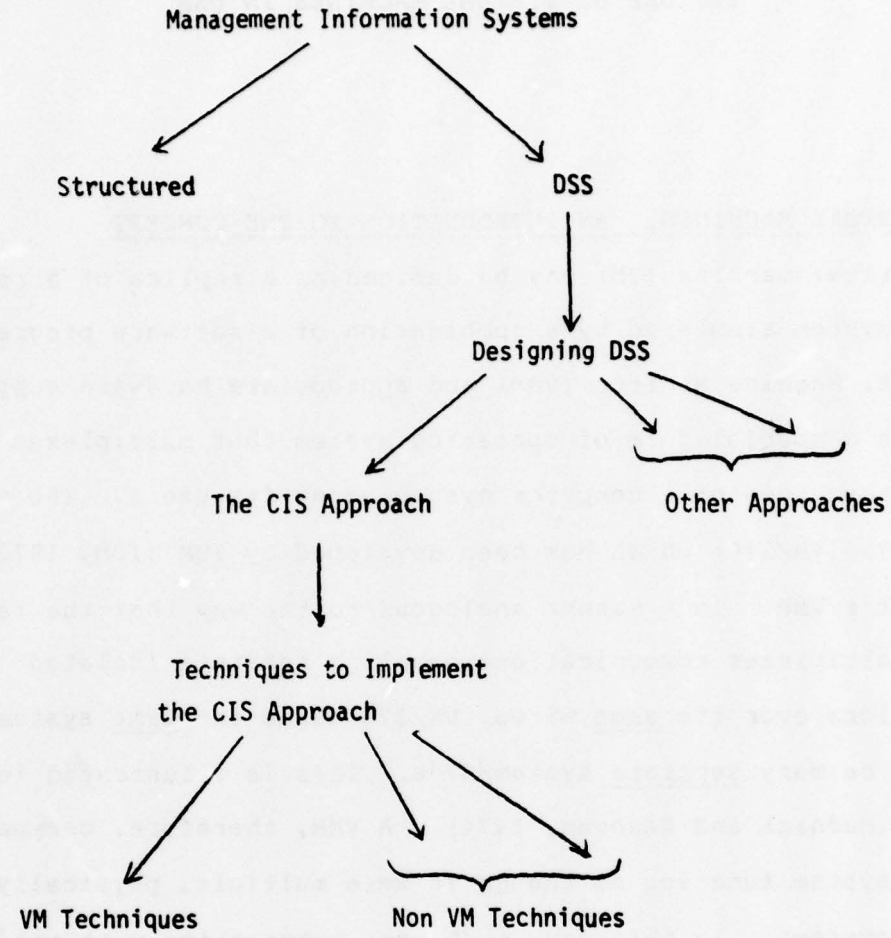

Management Information Systems
Structured
DSS
Designing DSS
The CIS Approach
Other Approaches
Techniques to Implement
the CIS Approach
VM Techniques
Non VM Techniques

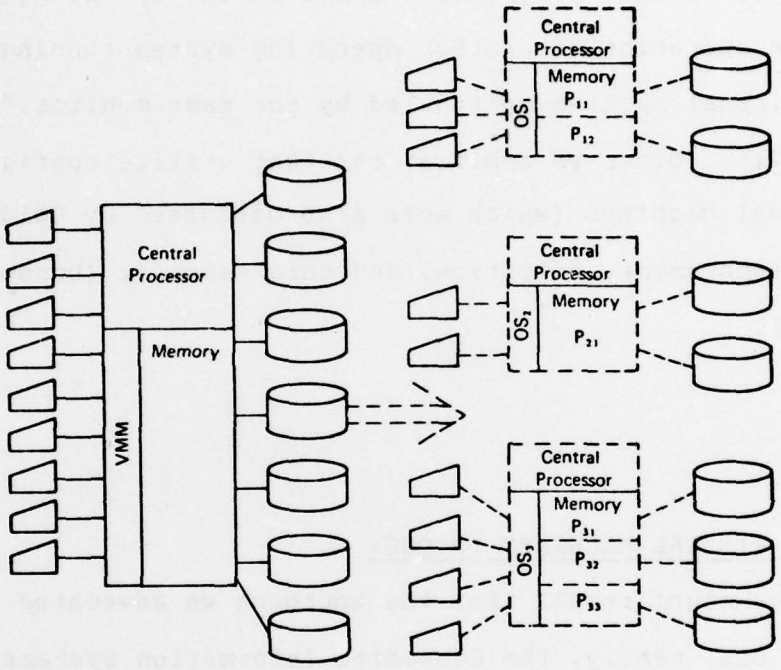
Figure I.2

II. USE OF VIRTUAL MACHINES IN DSS

II.1. VIRTUAL MACHINES: AN INTRODUCTION TO THE CONCEPT

A virtual machine (VM) may be defined as a replica of a real computer system simulated by a combination of a software program called the Virtual Machine Monitor (VMM) and appropriate hardware support. The VMM is a special form of operating system that multiplexes the physical resources of a computer system among its users. The virtual Machine/370 (VM/370) which has been developed by IBM (IBM, 1972) is an example of a VMM. In a manner analogous to the way that the telephone company multiplexes communications enabling separate isolated conversations over the same wires, VM/370 makes the same system/370 appear to be many separate system/370s. This is illustrated in Figure (II.1.) (Madnick and Donovan, 1974). A VMM, therefore, can make one computer system function as though it were multiple, physically isolated systems. In this way, each user interacting with one of the virtual machines, appears to have his own 370 computer. Thus, each user can select the operating system (e.g. OS/360, DOS, ... etc) of his choice to run on his "private" computer. This fact is depicted by the operating systems OS1, OS2, and OS3 in Figure (II.1.).

The figure also depicts the absence of communication among the seperate virtual machines. That is, each simulated machine is autonomous. The original philosophy of the VM concept was isolation ... each virtual machine is unaware that other VMs exist. This was perfectly acceptable, and sometimes even desirable or essential, for



(a) A real System/370 (b) Multiple Virtual System/370s

Figure II.1 Real and Virtual Machines

the type of applications in which VMs were first utilized. For example, VMs were (and still are) used to enhance software reliability. In this application area the most important aspect of a virtual machine system is precisely "...the high degree of isolation that a virtual machine monitor provides for each virtual machine operating under its control. In particular, programming error in one operating system will not affect the operation of another operating system running on an independent virtual machine controlled by the same monitor." (Goldberg, 1974). Other VM applications that utilize configurations of isolated virtual machines (which were also discussed by Goldgerg) are installation management, education, and data security (Donovan and Madnick, 1976).

II.2. USE OF VIRTUAL MACHINES IN DSS:

The reader might recall that the approach we advocated (in Part I) for designing DSS, namely, the Composite Information Systems approach, necessitates the ability to accomodate different data base and analysis systems in one integrated framework. We also pointed out that in order to achieve this we first have to solve the problems of incompatibilities that may exist between the different components. We would like now to look more closely at these problems and try to demonstrate why and how virtual machines provide a viable solution.

In Figures (II.2.) and (II.3.) we attempt to demonstrate the two basic types of incompatibilities that might exist. Figure (II.2.) illustrates the case where we have incompatibilities between the

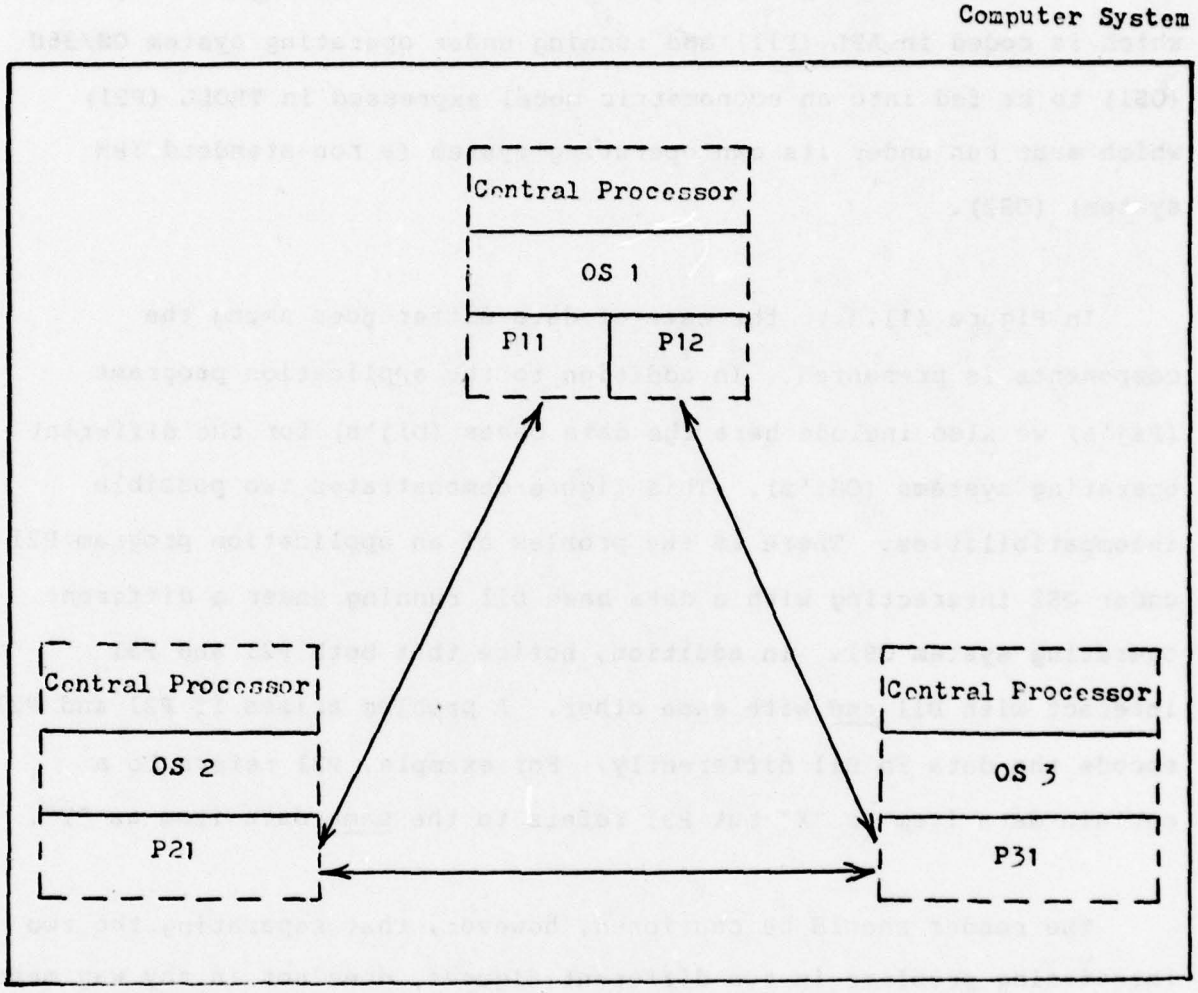

Computer System
Central Processor
OS 1
P11
P12
Central Processor
OS 2
P21
Central Processor
OS 3
P31

Figure II.2.

operating environments. The figure illustrates "a" possible configuration, where a set of different application programs (Pij's) operating in different operating environments (OSi's) need to interact with each other. For example, a user needs the output of his LP model which is coded in APL (P11) and running under operating system OS/360 (OS1) to be fed into an econometric model expressed in TROLL (P21) which must run under its own operating system (a non-standard IBM system) (OS2).

In Figure (II.3.) the case of data differences among the components is presented. In addition to the application programs (Pij's) we also include here the data bases (Dij's) for the different operating systems (OSi's). This figure demonstrates two possible incompatibilities. There is the problem of an application program P21 under OS2 interacting with a data base D11 running under a different operating system OS1. In addition, notice that both P21 and P31 interact with D11 and with each other. A problem arises if P21 and P31 encode the data in D11 differently. For example, P21 refers to a certain data item as "X" but P31 refers to the same data item as "Y".

The reader should be cautioned, however, that separating the two interfacing problems in two different figures, does not in any way mean that they are mutually exclusive. For, it is more likely than not, that both types of incompatibilities will exist simultaneously. It is thus, just for the purpose of simplifying our analysis that we represent them separately.

A closer look at both Figures (II.2.) and (II.3.) would show

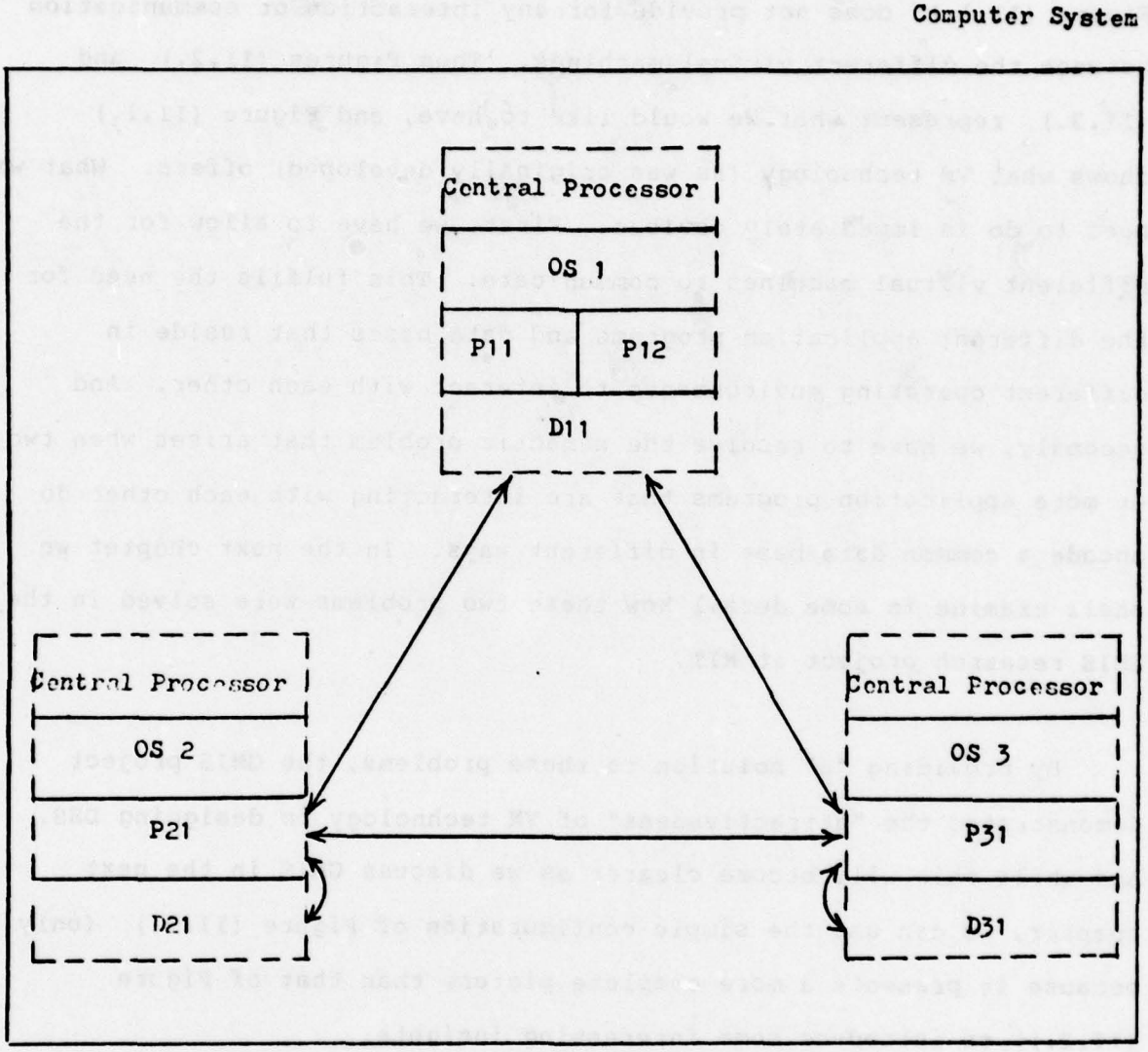

Computer System
Central Processor
OS 1
P11
P12
D11
Central Processor
OS 2
P21
D21
Central Processor
OS 3
P31
D31

Figure II.3.

that both have a close resemblence to Figure (II.1.). Indeed, the only difference is that the configuration of virtual machines illustrated in Figure (II.1.) does not provide for any interaction or communication between the different virtual machines. Thus Figures (II.2.) and (II.3.) represent what we would like to have, and Figure (II.1.) shows what VM technology (as was originally developed) offers. What we need to do is immediately obvious. First, we have to allow for the different virtual machines to communicate. This fulfils the need for the different application programs and data bases that reside in different operating environments to interact with each other. And secondly, we have to resolve the semantic problem that arises when two or more application programs that are interacting with each other do encode a common data base in different ways. In the next chapter we shall examine in some detail how these two problems were solved in the GMIS research project at MIT.

By providing "a" solution to these problems, the GMIS project demonstrated the "attractiveness" of VM technology in designing DSS. And while this will become clearer as we discuss GMIS in the next chapter, we can use the simple configuration of Figure (II.3.) (only because it presents a more complete picture than that of Figure (II.2.)) to introduce some interesting insights.

Consider a user working on a configuration of virtual machines as the one shown. He is, therefore, able to access any modeling or data base machine, or any combination of modeling machines connected to a data base machine, by specifying (through a mechanism of a virtual machine that manages the process) which machines are to be

interconnected. The system, then, provides him with the following "attractive" features:

1. Since each VM may run any existing model or program under its normal operating system, such a configuration eliminates the need to devote resources to translating application packages and programs between operating systems.

2. It permits interaction between application languages and programs not originally envisioned by their developers. For example, an analytical package could be greatly enhanced by having its data management capabilities extended. In ad-hoc Decision Support Systems this is particularly helpful where answers may be needed quickly, and there is often no time to transport the appropriate data bases and models to a common system.

3. The user is not restricted to a set of modeling or analytical capabilities that run on a particular operating system. Lifting such a restriction allows the user to access the modeling and/or analytical capabilities with which he/she is most familiar. This, of course, minimizes the user's need to learn new techniques.

With these insights into the plausibility of virtual machine technology in designing Decision Support Systems, we would like next to present a detailed discussion of a DSS that was developed at MIT and in which virtual machine techniques were used.

III. GMIS: AN EXAMPLE OF AN EXPERIMENTAL SYSTEM

In this part we shall discuss GMIS (the Generalized Management Information System). It is an example of a DSS developed using the Composite Information Systems (CIS) approach and employing virtual machines extensively (Donovan and Jacoby, 1975).

GMIS was developed at the MIT Energy Laboratory in conjunction with the Sloan School's Center for Information Systems Research and IBM. The project started in 1973 based on ongoing research in the Sloan School on file systems (Madnick, 1970) and operating systems (Donovan, 1972) (Madnick and Donovan, 1974). However, it has been the urgency of particular applications to energy problems that has shaped the work and quickened its pace.

During the energy crisis of the winter of 1973-74, policymakers in New England were handicapped by a lack of information about the region's energy economy. In response to this circumstance, the New England Regional Commission initiated a project to develop a New England Management Information System. The initial plan was to develop a "crisis management" system to assist in the handling of fuel oil allocation, but over time (although the original function remained an important one) the needs grew and the emphasis shifted. Problems of the economic impact of high oil prices took on more importance along with policies and programs to foster energy conservation. New issues rose concerning the location of major energy facilities, bringing a

need for analysis of associated economic and environmental issues.

Growing experience with the data also brought more demands on the system design. The data were of varying quality, and the data collection procedures were changing over time. (Series were being dropped and added and definitions were being revised.) Also, the requirements for protection have proven to be complex. They, for one thing, varied with levels of aggregation and time. (For example, an oil company might be willing to give out data on its aggregate transactions, but not on details that may help a competitor.) And finally, the need for a facility to apply various analytical models to the data became more apparent.

Given these circumstances, the research team's approach was to develop a general set of tools for speedy construction and easy modification of management information systems. It was recognized that the need is for a software facility for situations where the problem addressed is constantly changing and where the users are unable to specify exactly what to do, or precisely what the data streams will look like in the future.

To meet these requirements, certain characteristics of the system seemed essential. It had to be capable of storing, validating, and retrieving data and to have the capabilty to respond to changing data and data structure, and to varying protection requirements. It also had to provide tools for constructing analytical and statistical models that were to be applied to the data. However, it was recognized that building a facility that constructs these models from scratch would be

inefficient. It was conceived that a more efficient approach would be to build a facility that is capable of accessing models and packages that have already been developed and on which the users were already trained (Donovan and Jacoby, 1975).

In response to these technical requirements, the CIS framework was conceived as an appropriate and effective model and was used to design a prototype facility called the Generalized Management Information System (GMIS). The emphasis was on finding techniques for accomodating different data base systems and analysis systems in one integrated framework. Rather than force the conversion and transport of application systems to one operating system, a configuration of virtual machines capable of accomodating and integrating the different data base and analysis systems was developed. The configuration of virtual machines that was developed is depicted in Figure (III.1.) It was implemented on an IBM VM/370 virtual machine system (Donovan, 1976). In the sections that follow, we give an overview of the system's architecture.

III.1. COMMUNICATION MECHANISMS BETWEEN VMs:

We mentioned earlier that the original philosophy of the VM concept was isolation, that is, each virtual machine should be unaware that other VMs exist. Until recently, applications of VM technologies were consistent with this philosophy. Fortunately, with respect to technologies needed for Decision Support Systems, new mechanisms to facilitate communication have been developed. These include: the page

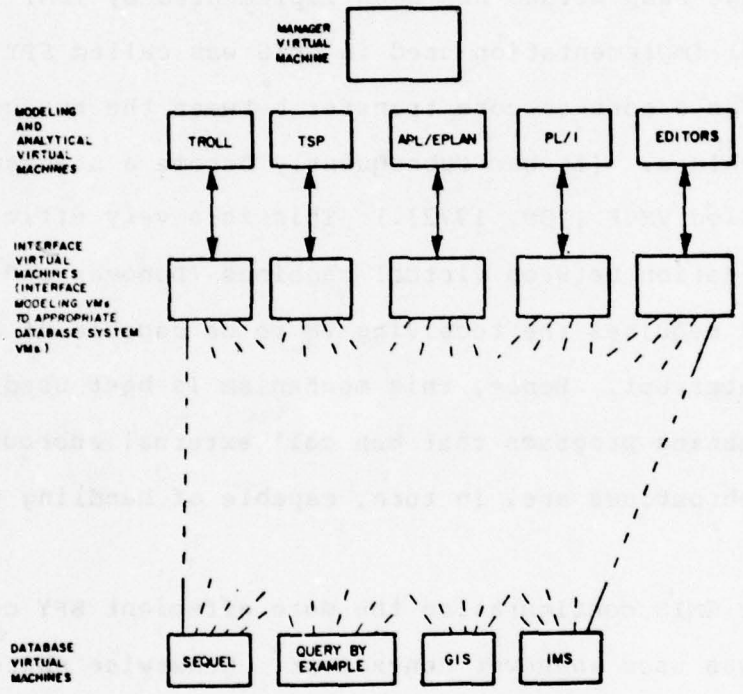

MANAGER VIRTUAL MACHINE
MODELING AND ANALYTICAL VIRTUAL MACHINES
TROLL
TSP
APL/EPLAN
PL/1
EDITORS
INTERFACE VIRTUAL MACHINES (INTERFACE MODELING VMs TO APPROPRIATE DATABASE SYSTEM VMs)
DATABASE VIRTUAL MACHINES
SEQUEL
QUERY BY EXAMPLE
GIS
IMS

Figure III.1.

swap method and the data move method (Astrahan et al, 1976) (Hsieh, 1974) segment sharing (Gray and Watson, 1975) channel-to-channel adaptor and virtual card punch and reader (Donovan and Jacoby, 1975) (Technical Reports No. 1 and No. 2).

The page swap method has been implemented by IBM. The experimental implementation used in GMIS was called SPY and can be thought of as a core-to-core transfer between the two communicating virtual machines. (It has subsequently become a standard VM/370 feature called VMCF {IBM, 1972}.) This is a very efficient mechanism for communication between virtual machines (Donovan and Jacoby, 1976). However, it requires the receiving VM to be capable of handling an external interrupt. Hence, this mechanism is best used between virtual machines running programs that can call external subroutines, where the external subroutines are, in turn, capable of handling the interrupt.

In the GMIS configuration the more effecient SPY communication mechanism was used whenever convenient. Otherwise shared minidisks were used. Let us discuss which mechanism was used, where, and why.

Figure (III.2.) depicts one instance of the general schema of Figure (III.1.), making explicit the communication mechanisms used. Two user analytical virtual machines are depicted, one that is executing PL/1 programs, the other executing APL programs, and both connected to a third data base machine. In the general schema a user logs into his analytical machine and sends a message to the virtual machine manager, which itself is being executed in a separate virtual machine. The message specifies what data base virtual machine the user

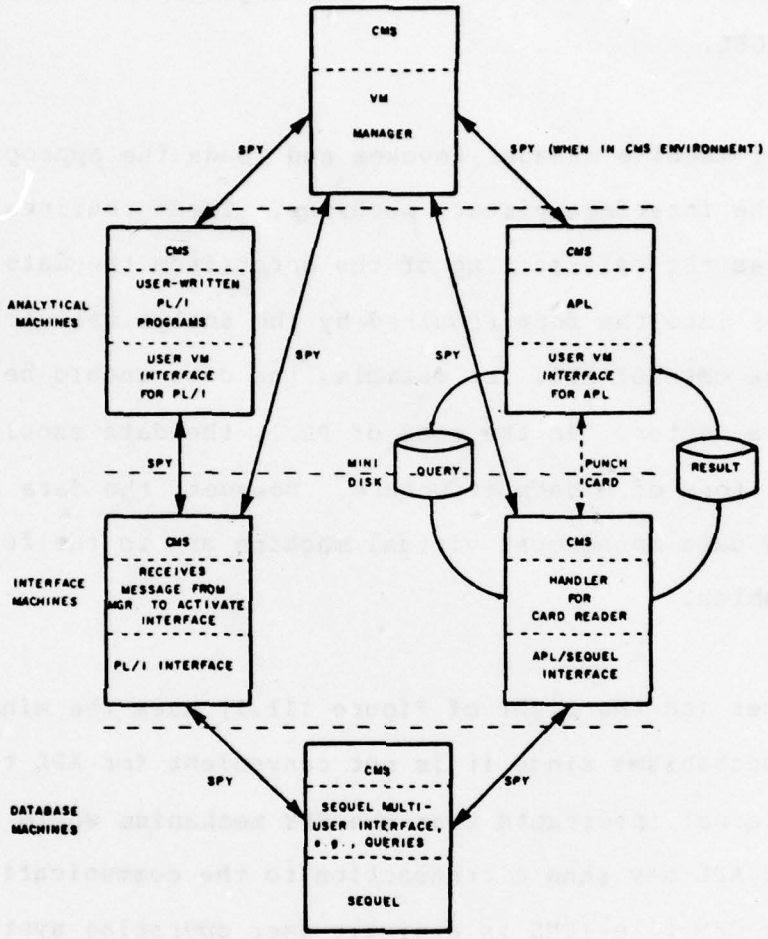

CMS
VM
MANAGER
SPY
SPY (WHEN IN CMS ENVIRONMENT)
ANALYTICAL MACHINES
CMS
USER-WRITTEN PL/I PROGRAMS
USER VM INTERFACE FOR PL/I
CMS
APL
USER VM INTERFACE FOR APL
SPY
SPY
SPY
MINI DISK
QUERY
PUNCH CARD
RESULT
INTERFACE MACHINES
CMS
RECEIVES MESSAGE FROM MGR TO ACTIVATE INTERFACE
PL/I INTERFACE
CMS
HANDLER FOR CARD READER
APL/SEQUEL INTERFACE
SPY
SPY
DATABASE MACHINES
CMS
SEQUEL MULTI-USER INTERFACE, e.g., QUERIES
SEQUEL

Figure III.2.

wishes to access. In the particular instance depicted in Figure (III.2.), both users have requested the same data base virtual machine, which in that case is a virtual machine running the relational data base system SEQUEL.

The virtual machine manager invokes and loads the appropriate routines into the interface virtual machines. These routines perform such functions as the reformatting of the data (from the data base virtual machine) into the form required by the analytical virtual machine. In the case of APL, for example, the data should be returned in the form of a vector. In the case of PL/1, the data should be returned in the form of a data structure. However, the data as stored in the depicted data management virtual machine are in the form of relations or tables.

The APL user (on the right of Figure III.2) uses the minidisk communication mechanisms since it is not convenient for APL to handle the special external interrupts that the SPY mechanism would require. The user VM for APL may send a transaction to the communications VM by writing it to a CMS file (CMS is a single user operating system commonly run on VM/370 {IBM, 1976}) and spooling a card from its virtual card punch to the communications VM's virtual card reader that generates a standard I/O interrupt. The communications VM is alerted to the user's request by the interrupt, reads the transaction from the CMS file, reformats it for the SEQUEL data base system, and sends the transaction to the SEQUEL VM via the SPY mechanism. After processing the transaction, the SEQUEL VM sends the reply to the communications VM via SPY, the communications VM reformats the reply for APL, writes the

reply to a CMS file, and signals the user VM running APL that the transaction is complete by spooling a card to its virtual card reader. The user VM may now read the reply from its CMS file and process it in any manner desired. This entire sequence is illustrated on the righthand side of Figure (III.2.).

A. Communication between the user VM and the Manager VM:

Since some analytical and modeling software facilities would be difficult to modify for communication directly with the manager VM, a seperate communication program (running under CMS) is invoked before the desired facility is activated. This program sends the necessary messages to the manager VM. The user may then activate an analytical or modeling facility under CMS or another operating system. That is, when a user first logs in, he runs under one operating system (CMS) and after communicating his needs with the manager VM, he loads the desired operating system into his VM.

B. Communication between the user VM and the interface VM:

For PL/1 and many other modeling facilities running under CMS, the communication to the interface machine is via SPY. However, for systems like APL and TROLL that run under their own environments, communication is via minidisks, since standard versions of these systems do not have the capability to handle the special SPY external interrupts but they are able to read and write disks, as well as punch and read cards. The message is written on a shared minidisk. The interface VM is notified that such a message is waiting by punching a card on a virtual card reader. The interface VM that has been in wait state reads that card and then reads the message on the minidisk.

C. Communication between the Interface VM and the Data Base VM:

SPY is used when the data base VM is running in a CMS environment (e.g. in the case of SEQUEL). However, communication is via minidisk, virtual card readers, and punches for data base systems that do not run in a CMS environment (as would be the case with IMS).

III.2. FUNCTIONS OF THE VIRTUAL MACHINES:

A. Functions of the Manager VM:

The primary function of the manager VM is responding to user requests for creating the connections between the virtual machines (by activating the interface virtual machine and data base virtual machine). The other function of the manager is to disconnect and automatically log out the appropriate interface VMs and data base VMs once the user has finished with them.

B. Functions of the Data Base VM:

The data base VM executes programs concerned with storing and accessing data as well as storing the data itself. Any data base system may run in such a machine. The systems primarily used for GMIS are interactive relational-type data base management systems (e.g. SEQUEL). The relational systems allow data base transactions to be entered on line, and prepare replies to these transactions in the form of single-valued results or tabular reports.

The processing scheme provides a multiple-user environment for each data base VM. Also, GMIS supports multiple data base VMs, each

processing transactions against a different physical data base, as shown in Figure (III.1.)

C. Functions of the Interface VM:

The interface VMs provide mechanisms for user VMs to interface with data base VMs. When a user virtual machine signals the manager VM to activate a configuration of virtual machines, this user VM indicates in which modeling or analytical environment it is currently running, and to which data base VM it wished to send transactions. The manager VM uses this information to signal an interface VM to load the appropriate interface for the particular user environment data base system combination desired.

III.3. GMIS AS A DECISION SUPPORT SYSTEM:

The GMIS configuration discussed above provides many of the capabilities needed to meet the requirements of Decision Support Systems outlined in section I.1.B.

GMIS provides the user with access to an interactive data management system SEQUEL. The relational system provides particular advantages for Decision Support Systems. Viewing data in the form of a table (relation) is conceptually simple (Donovan, 1976). Furthermore, the structure of the data, the ways a user will access it, and addition and deletion of data all change frequently in such applications. The SEQUEL relational system provides mechanisms for facilitating these changes, thus eliminating the need to define a more complex data

structure.

Decision Support Systems have requirements not only for data manipulation but also for facilities for data analysis. Systems like TROLL, TSP, and APL/EPLAN are good data analysis facilities but have poor data base facilities. Facilities like SEQUEL, IMS, ...etc, have good data base capabilities but poor analytical or statistical capabilities. The implementation of data base systems in the particular VM environment of GMIS allows the enhancement of any data management facility by extending its analytical and statistical capabilities at minimal cost. This enhancement is accomplished by running additional analytical systems which communicate with the data base machine.

A common requirement of a Decision Support System is to allow different groups working on similar problems access to the same data base. Each group, however, may be familiar with a different analytical system. The GMIS-type VM configuration allows different users (each using different analytical systems) to access the same data management system.

Another requirement of Decision Support Systems results from the fact that many data series needed by the decision-maker may be maintained in several different data management systems, and there is often no time to transport these data series to a common data management system. The GMIS configuration allows many data management systems to exist simultaneously. Any user or analytical system can access data stored in a variety of data systems.

In decision support applications it is often desirable for different (and often incompatible) application programs (e.g. models) to be able to interact with each other. For example there may exist an operational national supply model for natural gas consumption in the United States. At the same time, there may exist a regional demand for energy by sector. If a decision-maker wishes to study supply and demand of natural gas in New England, he may find it helpful to use these two models. The output of the first model (forcasted supply for the region) would be compared to the sum of output of the second model (demand by sector). Interactions of each model would then be performed until a balance occurred. However, the first model is written in TROLL, which operates under its own operating system and thus cannot be run on the same system as the second model, which is written in FORTRAN and running under OS. By bringing the two models up on the GMIS configuration (where each could access data generated from the other), their interactions would be facilitated through a common data management system.

And finally, an additional advantage of the GMIS approach is the increased security among the users of such a system. This increased reliability of GMIS is discussed elsewhere (Donovan and Madnick, 1975) and is an intuitive result of the fact that malicious or unintentional violations by the user must not only subvert the protection mechanisms of the operating system under which it is running, but also must subvert the protection mechanisms of the virtual machine monitor (VMM) if these violations are to affect another user. Hence this hierarchical protection mechanism can provide much higher security.

IV. AN EXAMPLE OF A GMIS APPLICATION

GMIS has been used extensively as part of the New England Management Information System (NEEMIS) project to provide decision support to regional energy policy-makers on a wide array of energy related issues. One application of GMIS in the NEEMIS project was the development of a prototype institutional DSS for energy conservation monitoring and analysis which was instrumental in attaining a significant energy saving in the New England states. All six states felt energy conservation within state-owned buildings was a desirable place to start a major conservation effort for four reasons:

1. Large dollar savings possible.
2. Federal funds were available to the states for such efforts.
3. Provide an example for the private sector to follow.
4. Reduction in future energy costs.

While the above are compelling reasons for conservation in state buildings, how does the policymaker make that happen? A two-pronged approach was developed: (1) Conservation monitoring of state buildings --- that is, monitor the monthly consumption in all buildings to allow cursory analysis making explicit which buildings are most likely candidates for savings --- and (2) Detailed analysis of buildings that appear to be large or disproportionately large consumers of energy to determine what changes, behavioral or structural, should be instituted.

Hence it was decided that an institutional DSS for long-term monitoring and analysis of state buildings should be constructed. It is important, however, to note the constraints that were placed on the system, the major one being time. The effort was started in response to Connecticut Governor Ella Grasso's mandate: Reduce consumption this winter (1975-76) in state facilities by 5 percent. That mandate was made at the beginning of the winter season.

Consider what that mandate means as far as implementation of such a system is concerned. How large an effort is required to build such a monitoring and analysis system?

Considering only the monitoring and cursory analysis portion, the system must contain information on building characteristics (location, floor space, type of usage, percentage of each type of energy source used for heating), vendor characteristics (location, type of energy supplied), supplier relationships (which vendors supply what to which buildings) and deliveries made (date, vendor, building, amount supplied, type of energy).

In addition, various supplemental information was needed, such as administrative hierarchy information about the various state agencies and organizations, such as the state police department --- its characteristics, budget, subordinate agencies, buildings owned --- and weather data to normalize for weather conditions by location and date. The incoming data must be screened and validated for reasonableness and completeness, then incorporated into the comulative data base.

For cursary analysis, it was necessary to produce both regular executive reports, such as monthly consumption by fuel type aggregated by major departments making explicit comparison with prior year's consumption adjusted for weather, and on-demand reports, such as total state energy expenditure during the first five months of a year or a list of residual fuel oil suppliers to the state.

Based upon estimates from conventional inventory control systems, if traditional system building approaches were used (writing a collection of FORTRAN, COBOL or PL/1 programs), many months --- if not years --- would be needed to construt a system adequate to meet merely the monitoring and cursory analysis requirements. The governor's mandate precluded this approach.

Furthermore, we have been assuming that the problem was well enough understood to allow the immediate specification of all inputs, outputs and algorithms necessary. Otherwise, the system would have to be revised, possibly many times, resulting in even more delay before becoming operational. In fact, it was found that even for what appeared to be the relatively straightforward task of monitoring the reporting of monthly consumption, major changes had to be made in the reports, type of data gathered, sources of that data and the computations.

For example, it quickly became apparent that reports which simply listed consumption by month for a particular building were not of enough assistance to the state energy officers in determinig which buildings they should focus efforts on. Reports that provided direct

comparisons between buildings such as depicted in Figures (IV.1.a) and (IV.1.b) were found to be more helpful. Further, once the monitoring system became operational, the way data was accessed and presented needed to be changed.

The states requested reports not only on individual buildings but by departments and other administrative entities that consist of several buildings. During that period, collecting accurate and timely consumption data from the end customer was found to be both difficult and time consuming. An alternative and much more satisfactory mechanism of data gathering was found by obtaining information directly from the energy suppliers to the state buildings.

Notice that many changes in the data needed, the data aquisition procedures and uses and reports necessary occured after the (prototype) Decision Support System was put into use. If a prototype had not been built, the need for this change probably would not have been realized until the full-scale operational DSS had been constructed many months later, necessitating major redesign resulting in further delays.

The GMIS-type consumption monitoring and cursory analysis system developed used a configuration of virtual machines that included the analytical languages APL and EPLAN and the data management system SEQUEL.

SEQUEL is a relational data base system which provides considerable flexibility and allows access to the data under multiple and different criteria. APL/EPLAN provided an interactive analytical

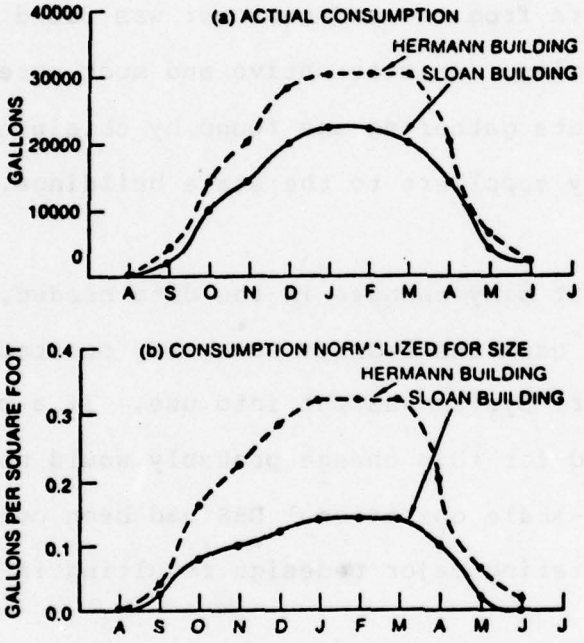

(a) ACTUAL CONSUMPTION
HERMANN BUILDING
SLOAN BUILDING
GALLONS
40000
30000
20000
10000
0
A S O N D J F M A M J J
(b) CONSUMPTION NORMALIZED FOR SIZE
HERMANN BUILDING
SLOAN BUILDING
GALLONS PER SQUARE FOOT
0.4
0.3
0.2
0.1
0.0
A S O N D J F M A M J J

Figure IV.1.

facility that facilitates complex operations on vectors and convenient plotting of results.

Note that neither APL/EPLAN nor SEQUEL separately would be adequate to serve the needs of monitoring and cursory analysis. Furthermore, these facilities cannot normally be combined since they are operationally incompatible, that is, they operate in different operating system environments. Combining these facilities, using GMIS, provided the basis for the prototype institutional conservation monitoring system.

This version of the conservation monitoring system was completed in time for use in the winter of 1975-76.

Now let us consider the construction of mechanisms to perform detailed building analysis. This was potentially an even more difficult problem, especially if traditional approaches had been used. Using the cursory analysis reports, the states were assisted in deciding on which buildings they should concentrate their efforts. However, the decision of what specific actions should be initiated is left unresolved. For example, an output from the cursory analysis system is displayed in Figure (IV.1.).

This graph depicts the monthly consumption of two buildings and the same information normalized for square footage. Note that the Hermann Building has significantly higher consumption, both in total consumption and consumption per foot; hence it is a likely candidate for conservation measures. What should be done? Add storm windows?

Change the heating system? Insulate?

Two approaches could have been taken. First, experiment with the building itself --- very costly. Secondly, build a computer model of the building and experiment with it --- significantly cheaper but would be very time consuming if the model was built from scratch. Hence if the second approach is taken, it is important to build upon the work of others. Three existing general building analysis models were available: NECAP (NASA's Energy Cost Analysis Program), CERL (Thermal Loads Analysis and Systems Simulation Program) and ECM (Energy Conservation Manual).

NECAP and CERL, an extension of NECAP, already existed as large-scale computerized models. (It has been estimated that more than $1.2 million was spent on the development of the current NECAP program and its earlier versions). ECM is a simplified analysis procedure. These systems are complementary and they can be used to validate each other. For these advantages to be effectively realized, however, the models should be integrated into one cohesive system.

Using the model integration capabilities of GMIS, the NECAP and CERL programs were transferred. The simpler ECM model was implemented by the project staff. All of these models required that certain additional information be added to the data base to perform detailed analysis of a building. Information needed for a NECAP analysis included: structure of building, types of heating and cooling units, ventilation, shading of building, number of days of sun, data for each exterior wall, door, window, roof and floor and schedules for people,

lighting and equipment usage.

The specification of the NECAP model for a particular building requires considerable more data on the characteristics of that building than the ECM model. For the NECAP model, as an example, seven man-days were required to gather the additional information for a large state medical laboratory. Although the resulting detailed NECAP model simulation for that building produced consumption results that were very close to actual energy consumption, the detailed data required for NECAP makes such a procedure difficult for analyzing a large number of buildings. The ECM procedure can be used for a much more aggregate level of analysis and requires data that can be fairly easily obtained. For most common types of buildings, ECM produces results that are also very close to actual energy consumption.

Using the model integration features of the GMIS technology, it was possible to have all three models available for use in the winter of 1975. Procedurally, the cursary analysis was used to focus attention on buildings that appear to offer high potential for energy conservation. The ECM model was used to experiment with many simple conservation options for the selected buildings. And for buildings that have unusual characteristics or are being considered for extensive changes, the NECAP model was used for more detailed and accurate analysis.

Hence, the ability to assimilate existing models, to assimilate data series and to change computations and structures of that data quickly were all important to the construction of this DSS. The GMIS

approach used facilitated this.

In the specific case of Governor Grasso's request to reduce energy consumption, the Connecticut Department of Planning and Energy Policy was able to meet and exceed the goal by reducing energy consumption by 7.5 percent (discounted for weather), a savings of more than $1 million.

The effectiveness of the prototype provided the basis for the implementation of a long-term, high-performance institutional DSS for Connecticut and the adoption of such a system by other New England states. Furthermore, there have been two additional benefits over direct energy saving to such a consumption analysis facility: It provided a basis for the states' eligibility for a portion of the $150 million in federal funds available under Public Law PL-94-163 for energy conservation and it allowed the states to take a leadership role and encourage the private sector to institute similar conservation programs.

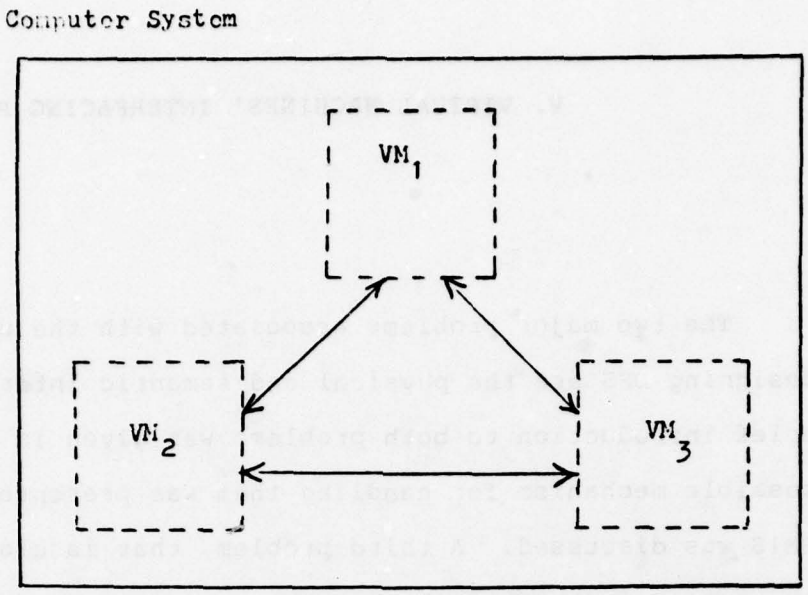
V. VIRTUAL MACHINES' INTERFACING PROBLEMS

The two major problems associated with the use of VM technology in designing DSS are the physical and semantic interface problems. A brief introduction to both problems was given in chapter II, and "a" possible mechanism for handling them was presented in chapter III, when GMIS was discussed. A third problem, that is closely related to the above two, is that of the high operational costs associated with a GMIS-type configuration.

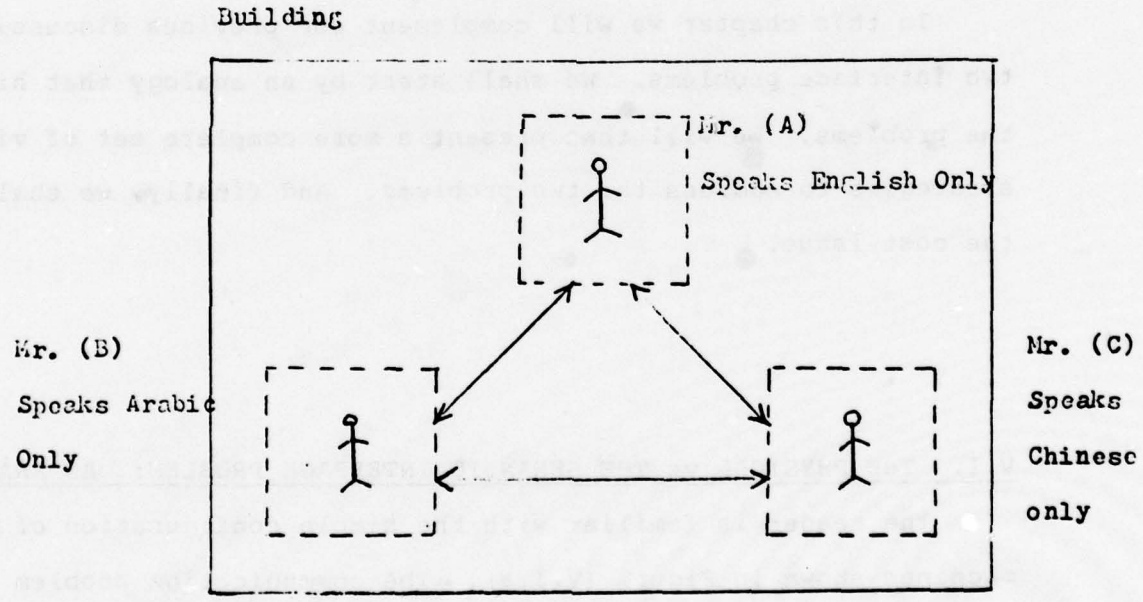
In this chapter we will complement our previous discussion on the two interface problems. We shall start by an analogy that highlights the problems. We will then present a more complete set of viable strategies to address the two problems. And finally, we shall discuss the cost issue.

V.1. THE PHYSICAL vs THE SEMANTIC INTERFACE PROBLEM: AN ANALOGY

The reader is familiar with the simple configuration of virtual machines shown in Figure (V.1.a). The communication problem between these three virtual machines (that are residing in the same computer system) is analogous to the communication problem between the three gentlemen of Figure (V.1.b). "A", "B", and "C" are residing in three different rooms of the same building, and each speaks a different language. How can these persons communicate?


Computer System
VM1
VM2
VM3

(a)


Building
Mr. (A)
Speaks English Only
Mr. (B)
Speaks Arabic
Only
Mr. (C)
Speaks
Chinese
only

(b)

Figure V.1.

The first step is to establish a means of physical communication. And this is relatively easy. It can be done, for example, by using a simple network of intercom lines. Thus, "A" can transmit a message in English to either "B" or "C" or to both. But, will they understand it? NO.

What we still need to add to our communication network is a mechanism that resolves the semantic problem that exists. And, our simple analogy can demonstrate how complex a task this can be. Consider a possible mechanism that performs the following operations:

1. Identify the message sender e.g. "A".
2. Identify the sender's language e.g. English.
3. Identify the message receiver e.g. "B".
4. Identify the receiver's language e.g. Arabic.
5. Select the suitable dictionary e.g. English-Arabic.
6. Translate message.
7. Send translated message.

This analogy serves three important purposes. Firstly, it explains what each type of interface means. Secondly, it demonstrates the clear advantage the VM technique provides in solving the physical interface problem. Being in the same building, the physical communication between the three men was easily established by a relatively simple intercom network. The task would undoubtedly have been considerably more difficult if each of the men was in a different building, and the buildings were served by different telephone systems. And thirdly, the analogy highlights the fact that the semantic problem

is far more complex and much less straight forward than the physical interface one.

V.2. PHYSICAL INTERFACING OF VMs IN A DSS

Strategies for interfacing the components in a virtual machine based DSS are explored in Technical Report No. 1. These strategies focus on the portal interfacing mechanisms i.e. getting data from an output port of a virtual machine to the input port of another. In particular, four major classes of VM portal interfacing strategies are developed.

The first type of strategies is direct vitualization of existing mechanisms on real machines. This includes virtual channel-to-channel communications between two machines, sharing of a disk between two machines, and telecommunications between two machines. The second type of strategies is virtualization of existing I/O devices and direct mapping between two virtual devices of two virtual machines. This includes mapping of virtual card reader to virtual card punch, mapping of virtual tapes, and mapping of two similar virtual consoles. The third type of strategies is to extend the capabilities of the Virtual Machine Monitor (VMM) to facilitate indirect mapping of virtual I/O devices. This includes mapping of virtual printer output to a virtual card reader or a virtual console, and the mapping of virtual terminal output to another virtual terminal or to a virtual card reader. The fourth type of strategies is to extend the capabilities of the VMM to support direct inter-virtual machine communications. This includes

communications by virtual processor interrupts, specially coupled virtual machines, and shared writable memories.

V.3. SEMANTIC INTERFACING OF VMs IN A DSS

As was mentioned in section V.1., the semantic problem is the more complex problem we face in applying VM technology in the design of DSS. As a step towards "more effective" solutions to this problem, we introduced in Technical Report No. 5 the concept of the Composite Database System (COMPDATA). In particular, COMPDATA is intended to provide an easy-to-use interface for accessing data, and improves the joint usage of multiple and different data base systems.

Figure (V.2.) shows one possible scheme of using COMPDATA in a GMIS-type system. The Language Interface VM interacts with any User VM to provide a unified high-level language interface to all stored data in the system. The Semantic Data System VM maintains information on the user applications as well as information on the various data bases so that the Language Interface can interpret a user data request. A user data request is mapped into the appropriate operations on the virtual relations corresponding to the stored data. Each Database System Interface VM is responsible for realizing these operations on virtual relations by interacting with the Database VMs. Such a scheme, therefore, appears to the user as a single and easy to use data base system.

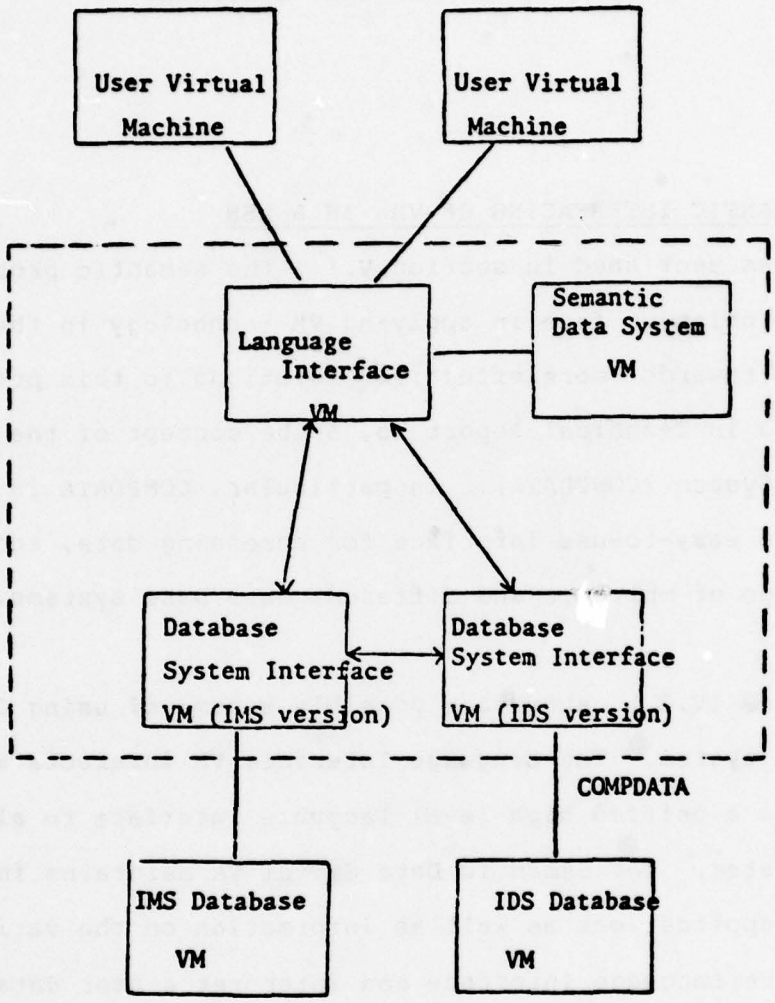


Figure 7.2. COMPDATA in GMIS-type system

V.4. VM's PERFORMANCE COSTS

In this section we discuss another type of problem associated with using VMs in developing DSS: Their operational costs.

There are additional costs involved in running a jobstream on a virtual machine instead of a real machine. These costs include the extra resources e.g. main memory and processor cycles needed by the Virtual Machine Monitor (VMM) --- and the potential drop in the system throughput which results. However, these costs can be more than compensated for, in certain applications, by the decrease in fixed costs of developing such applications. This is particularly true in the case of DSS.

To place these costs into a framework relative to traditional technologies, we refer to Figure (V.3.), where fixed costs (costs of developing a management information system) and variable costs (costs of operating such a system) are depicted. The units of the Y-axis are dollars; the units of the X-axis are time or number of queries made.

The dashed line represents costs typically found in traditional management information systems i.e. systems supporting structured type decisions. For example, in a payroll system the focus is on low variable costs (slope of line is small) as each check issued must cost only pennies even though the initial development costs may be quite high.

The "*" curve represents typical costs associated with the development and operation of an institutional Decision Support System

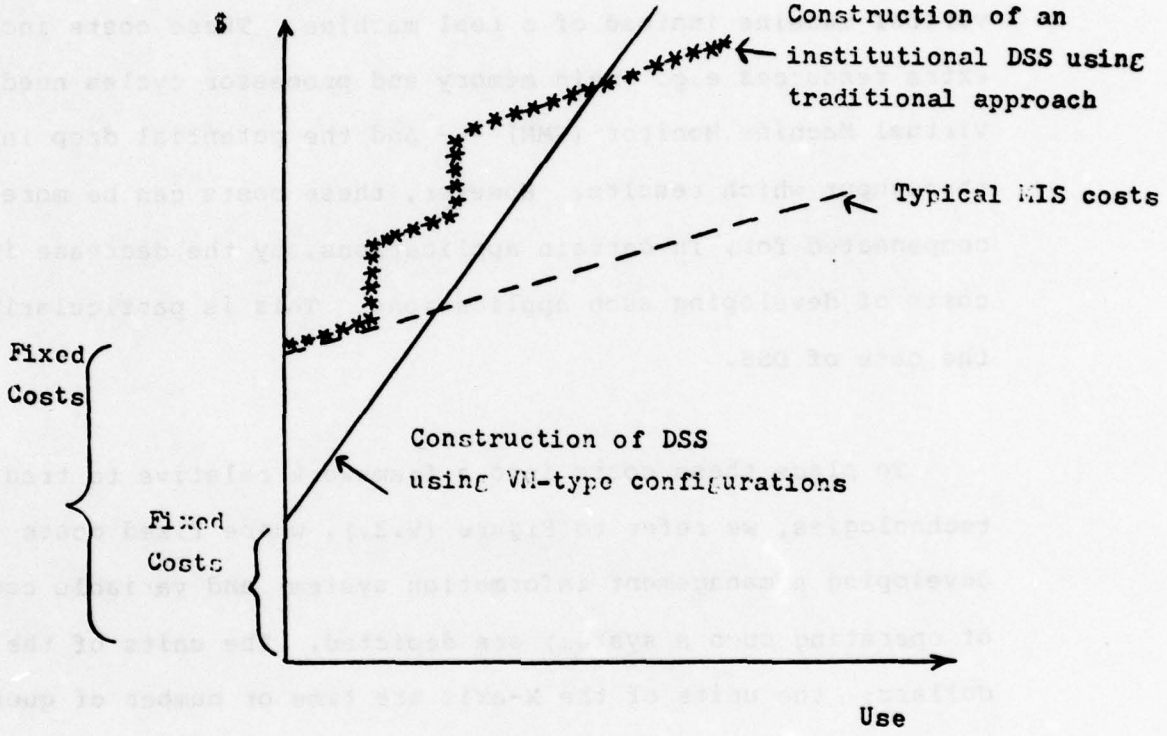

$
Construction of an
institutional DSS using
traditional approach
Typical MIS costs
Fixed
Costs
Construction of DSS
using VM-type configurations
Fixed
Costs
Use

Figure V.3.

using traditional technologies. For example, a portfolio management system often is first brought into operation for a short time only to find additional fixed costs must be incurred due to changes in perception of the function of the system or in available data. Typically, these changes become less frequent until finally stabilizing, and attention is then given to tuning the system (reducing the slope of the curve).

The solid line depicts a more desirable cost curve for a DSS (both ad-hoc and prototype development of institutional) as they are seldom in operation long enough for the break-even point (intersection with the dashed line) to be reached. Donovan and Jacoby (1976) reported that the technologies used in GMIS-type systems, do in fact allow for the lower fixed costs depicted in the solid curve, and that the total cost is less even if we take into account the higher variable costs (as depicted by the larger slope of the larger slope of the solid line of Figure (V.3.)).

The higher variable costs in a GMIS-type DSS are due to two factors. First, and as was stated in the openning paragraph of this section, running a virtual machine is more costly than running a real one. In addition to that, in a GMIS-type DSS certain mechanisms are needed to resolve communication problems between the different virtual machines. These arise because of the incompatibilities between the operating environments and data bases. These software mechanisms, then, are envoked whenever an interaction between two or more different components is required. And this, of course, increases the costs of operation.

It is encouraging to mention, however, that the GMIS research team at MIT has successfully demonstrated the potential for significant performance improvements. In one of the experiments, the overhead costs were reduced by a factor of two. This result together with a more complete discussion of this topic are reported in Technical Report No. 2.

VI. CONCLUSION

This is the final report in the study of virtual machine technology as a tool in developing Decision Support Systems. We found that virual machines are especially suitable for building both ad-hoc DSS and the prototype phases for institutional DSS. This is a new application area for virtual machines, and where new technical problems have yet to be resolved. The two major problems, as was mentioned in this report (and in much more detail in reports No. 1 and No. 2) are those associated with the physical and semantic interfaces between different virtual machines. With respect to the first problem, further investigation of performance implications of the mechanisms used for resolving component incompatibilities is needed. The semantic problem, however, is the more difficult and for which much research effort is needed to arrive at better solutions.

The COMPDATA structure we discussed in chapter V (and in much more detail in Technical Report No. 5) can be used as a basis for further research in the semantic interface area. In particular, we suggest the following research topics: development of suitable semantic data models; development of semantic data dictionary systems; and automating the development of interface virual machines. This last research area is particularly interesting, as it would allow for the rapid incorporation of new data data bases, thus facilitating the development of GMIS-type systems.

PRECEDING PAGE BLANK

Addendum Technical Report 1

Use of Virtual Machines for Development of Decision Support Systems : Strategies for Interfacing Virtual Machines

Chat-Yu Lam

Stuart E. Madnick

Strategies for Interfacing Virtual Machines

I. Motivation

Decision Support Systems (DSS) are a subset of Management Information Systems (MIS) which are intended to directly support management decision-making. An ad-hoc DSS (Donovan and Madnick 77) is concerned with aiding decisions that are often non-recurring, the time for analysis is short, and the nature of the decision as well as the approach for its analysis may change. Thus the databases and the tools needed for analysis of an ad-hoc decision cannot be anticipated, and once the analysis is completed and the decision is made, the particular combination of databases and tools may not be needed any more.

To be effective, an ad-hoc DSS must be flexible enough to assimilate different databases and tools rapidly and make them jointly usable to aid the decisions. Since the databases and tools are not known long in advance, and existing tools and databases must be used due to the urgency of the decision, this poses a serious interfacing problem for the designer of an ad-hoc DSS.

The problem is that these databases and tools are often developed independent of one another, and are often incompatible with one another. These tools and databases may reside on totally different machines and/or execution environments, and/or possess incompatible I/O ports, thus output of one cannot be easily used as input to another.

One approach to solve the interfacing problem is to make use of virtual machines (VM). Virtual machine technologies make possible different, incompatible execution environments and (virtualized) machines to coexist on one physical machine, thus effectively bringing together the tools and databases. It may be possible to take advantage of the virtualization schemes of the computer resources to solve the problem of communications and data transfer among the tools and database systems.

In this report, we clarify some concepts regarding tools, databases, and virtual machines. Then we explore various strategies that take advantage of virtual machine technologies for interfacing tools and database systems in an ad-hoc DSS.

II. Virtual machine concept and an example

A basic concept in virtual machine technology is that of mapping real computer resources into groups of virtual computer resources, each called a virtual machine (see Figure II.1).

The mapping of real resources into virtual resources is supervised by the virtual machine monitor (VMM). VMM takes control when a virtual machine tries to use a virtual resource that has no real correspondence. VMM then tries to establish the mapping or take appropriate action, such as simulation of the real resource, or enforce security violation actions. The particular scheme that automatically transfers control to the VMM is dependent on the particular implementation.

The operating system supervisor (OSS) manages all the virtual resources in a virtual machine as if they were real resources. Several programs may be in various states of execution on the virtual machine, each corresponds to a process. To effectively manage these processes and the virtual computer resources, OSS ensures that each process has a rightful claim to a domain of virtual resources. When a process attempts to access a virtual resource not in its domain, the OSS takes control and performs the appropriate actions. Again, the particular scheme that enables the OSS to take control under these situations is implementation dependent. These schemes correspond to the SYSTEM/USER states, or the rings of protection mechanisms in some

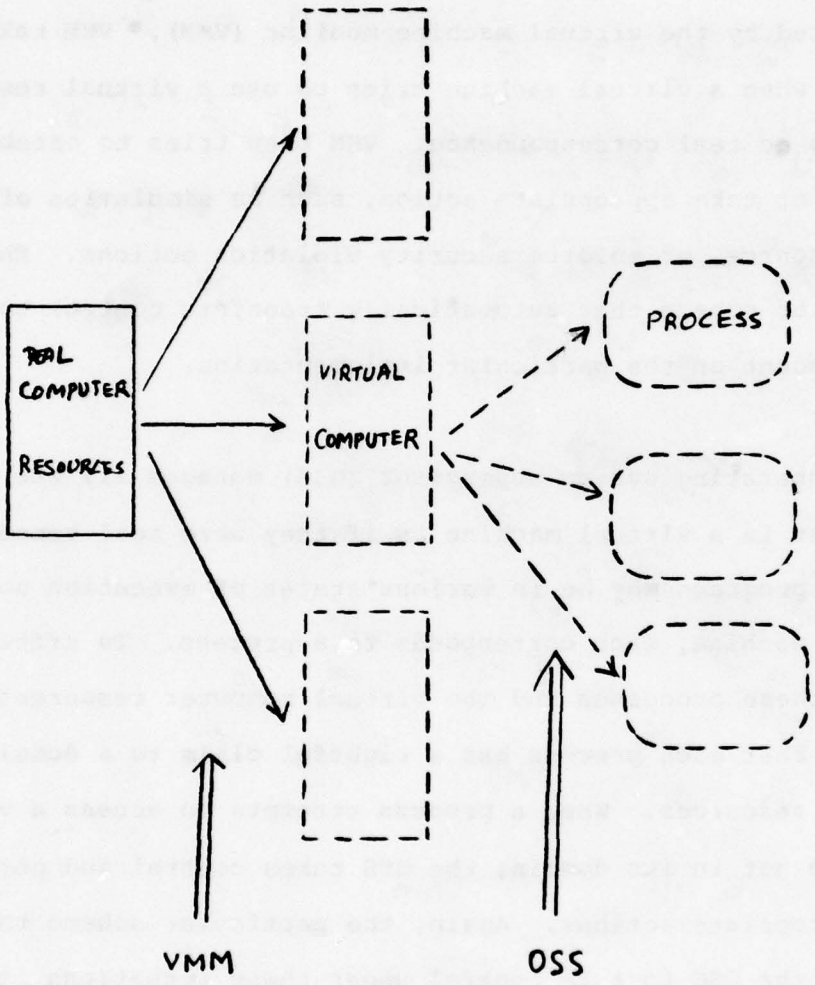


Figure II.1 VIRTUAL MACHINE CONCEPT

operating systems.

To illustrate the virtual machine concept, we use the IBM VM/370 as an example (IBM 77) (Figure II.2). On VM/370, each virtual machine is functionally equivalent to a real IBM System/370 computer, including all privileged instructions and I/O control. In actuality, each virtual machine executes in USER mode, while the VMM executes in SYSTEM mode. When a real interrupt occurs, either due to a program issuing a supervisor call (SVC), or a privileged instruction, or other events that require supervisory attention, the VMM takes control. VMM examines its status table for the virtual machine causing the interrupt. If the virtual machine is to be viewed as being in USER state, VMM reflects (i.e. simulates) the interrupt to the virtual machine's OSS, which will handle the interrupt. If the virtual machine is to be viewed as being in SYSTEM state, VMM handles the interrupt, which may involve simulation of priviledged or I/O instructions or performing some other supervisory functions.

Mapping of virtual to real resources on VM/370 is summaried briefly as follows. Each virtual machine has its own virtual memory. Mapping between real and virtual memory is facilitated by using system tables and the dynamic address translation (DAT) hardware. The system tables indicate the status and location of each block of virtual memory, whether it is in real memory, or on a paging device. All references to the virtual memory are

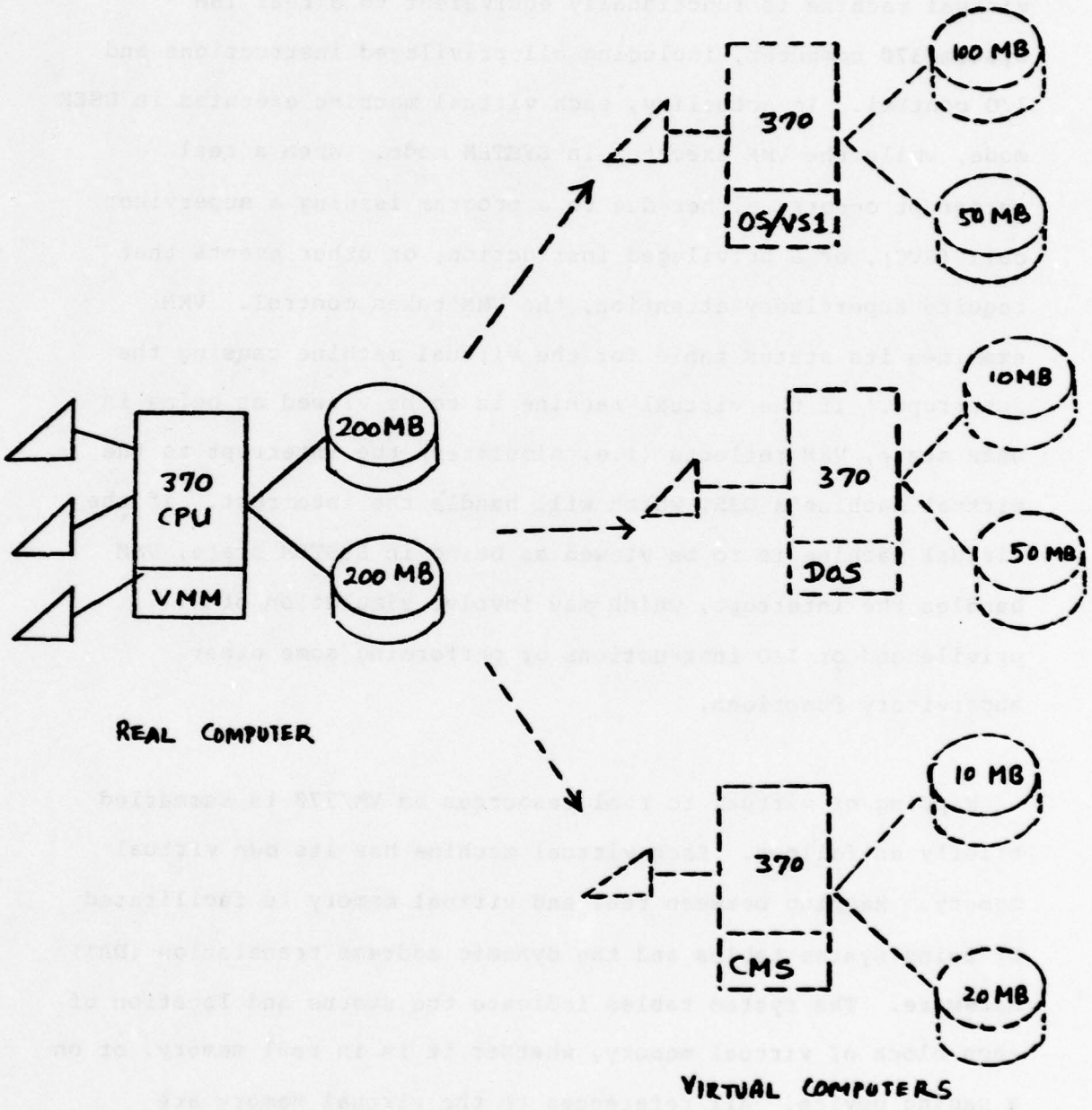


Figure II.2 VM/370

automatically translated by the DAT, using information in the system tables, into references to real memory. If a referenced block does not exist in real memory, the VMM will bring the block into real memory and update the system tables accordingly before completing the reference.

The processor is switched among the virtual machines, therefore each virtual machine appears to to have its own processor. Virtual tape units of a virtual machine are mapped into real tape units. A real transmission control unit (TCU) may be mapped into several virtual TCUs each having a subset of the lines on the real TCU. A virtual disk may have a real correspondence. A virtual disk may also be a subset of a real disk, i.e. a real disk may be mapped into several virtual disks, each having several cylinders of the real disk. Unit record devices such as printers, card readers, and card punches may be assigned physical counterparts or, more typically, be virtualized using SPOOL files. The contents of these files may be printed or punched out when a corresponding real device is available.

III. Components in a virtual machine based DSS

We can identify five general types of components in a virtual machine based DSS. These are: (1) the operating system execution environment, (2) generalized language interpreter execution environment, (3) special purpose language processor execution environment, (4) tools, and (5) database systems. These components are briefly discussed below.

III.1 Execution environment - operating system

An operating system supervisor manages the resources in a computer system. The OSS provides macros and subroutines that a program can invoke to utilize the computer resources, e.g. to read a file stored on disk. The OSS also provides special utility programs that an user can invoke to perform various maintenance functions, e.g. allocate space for a file, copy a file, etc. An user communicates his/her requests for service to the OSS via a control language. There is a component of the OSS that always listens for user requests. For example, in the case of a non-multiprogrammed OSS, the listener may pick up a request from a user to run program X. The OSS initiates program X and passes control of the central processor unit (CPU) to program X. When program X finishes its processing, control of the CPU is returned to OSS and the listener is reinitiated to wait for more user requests. We often refer to this OSS loop as a listening loop.

From the point of view of the user within the OSS listening loop, the OSS, its utilities, and the computer resources collectively provide him/her an execution environment to construct new programs, run existing programs and perform maintenance functions provided by the OSS. We refer to this type of execution environment loosely as an operating system.

III.2 Tools and special execution environments

A tool is a program that represents a well-defined, closed-ended computation. A tool usually receives control from a supervisor of the execution environment that the tool is in. It performs its computation and may carry on a dialog with the user during its computation, typically to obtain some input. At the end of the computation, the tool returns control to the execution environment supervisor.

There is another class of programs that behave quite differently from tools. We refer to this class of programs as special execution environments. A program of this type receives control from the execution environment supervisor, but does not immediately return control to the supervisor. In fact, the program proceeds to set up its own listening loop and in effect traps the user in another execution environment. In general, the user communicates with such a special execution environment via a special language (e.g., an APL interpreter). The user is able to construct new tools, run existing tools and may be able to

perform some maintenance functions within the special execution environment. Although eventually control is returned to the execution environment supervisor, by explicit user request, the computation represented by this program is ill-defined, and open-ended.

We have identified two major subclassses of special execution environments. These are the generalized language interpreters and the special purpose language processors.

(1) Generalized language interpreters

Generalized language interpreters include all the interpretive programming languages, e.g. APL, BASIC, and so forth. A generalized language interpreter usually is directly under an operating system. The listener of the execution environment is the language interpreter. Under this execution environment, the user is able to construct new tools interactively, run existing tools, and to perform some maintenance functions. These maintenance functions are often a subset of those available from the host operating system. There is considerable variations in the scope of these functions across different host operating systems. For example, under the Cambridge Monitor System (CMS) (IBM 77), an APL user can invoke most of the CMS commands via the APL shared-variable facility, however, under OS/VS1, the APL user

has very limited access to OS/VS1 system maintenance functions.

(2) Special language processors

Specialized language systems have been developed to reduce user programming efforts and to better suit the particular application interests of the users. For example, special systems have been deisgned for construction of simulation models, for construction of econometric models and for data analysis, and so forth. Many of these systems behave like a special execution enviornment we discussed above. Furthermore, due to the special application orientation of most of these systems, they often have their own special file systems particularly suitable for their applications. Thus, the maintenance functions available to the user in such a special execution environment may be quite different from those available to the user if he/she were in the host execution environment. Notice also that an execution environment may reside on another execution environment which may reside on yet another execution environment, and so forth. For example, Figure III.1 illustrates that a modified version of the Time Series Processor (TSP) runs as a special language processor under the Consistent System (Cambridge Project 74), which is a special language processor for social science research that runs under the MULTICS operating system on a Honeywell 6180 computer.

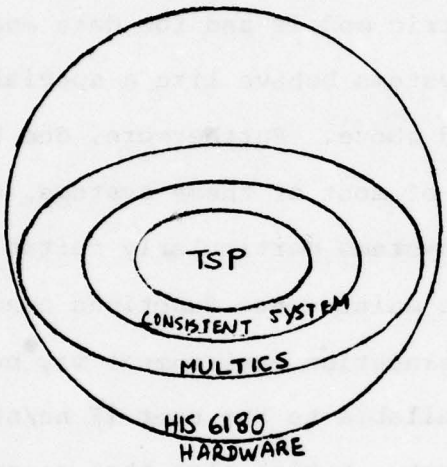


Figure III.1 TSP SPECIAL LANGUAGE PROCESSOR

III.3 Database systems

A database system (DBS) consists of two basic components: the databases and a program that manages these databases and provides data access interface(s) for the user of the databases. The databases may be constructed using very complex data structuring techniques, e.g. indexing, chaining, etc, or they may be constructed using very simple data structuring techniques, e.g. sequential file organizations. The data access interface, also referred to as the data manipulation language (DML), could be a set of subroutines usable only via program calls, or it may be a language interpreter that a user can use interactively. In either case, the nature of the language supported may be either low level operations such as GETNEXT RECORD, or high-level operations such as PRINT ALL EMPLOYEE WITH SALARY OVER 35000. From the point of view of the user, a DBS consists of a collection of logical databases, and the data access language(s) to manipulate these databases. This is represented diagrammatically in Figure III.2. The user may interact with the DBS via a procedural program or without using such a program, or both, depending on which mode(s) of operation that the DBS supports.

In some cases a DBS may be used as a tool. For example, a specific application program and the DBS software may be linked into one load module and executed as a giant program, like an ordinary tool. This situation is depicted in Figure III.3.

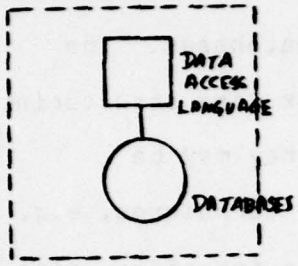


Figure III.2 SCHEMATIC OF A DATABASE SYSTEM

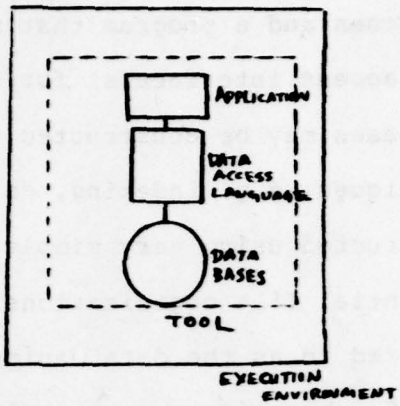


Figure III.3 DATABASE SYSTEM RUN AS A TOOL

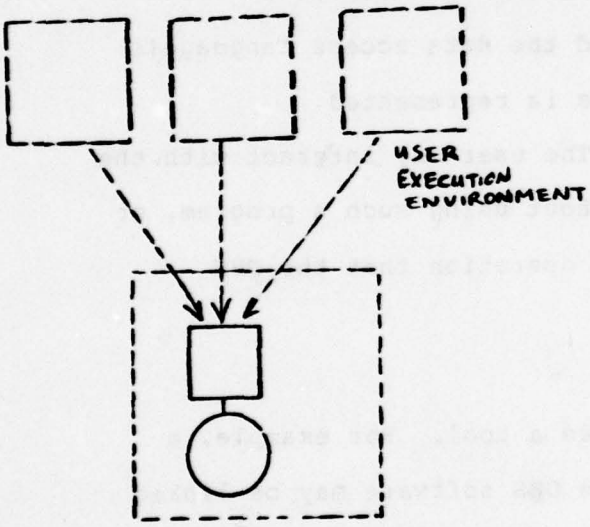


Figure III.4 DATABASE SYSTEM SERVING MULTIPLE USERS

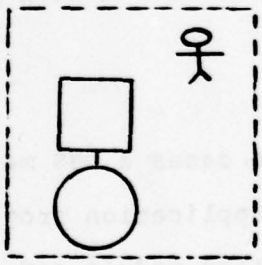


Figure III.5 DATABASE SYSTEM RUN AS AN EXECUTION ENVIRONMENT

Since a DBS is a very expensive resource and is often used to support all the data needs of an information system, it may be desirable to execute the DBS independent of other application programs. The DBS executes like an execution environment servicing database requests from various users. This situation is depicted in Figure III.4. Yet another mode of operation for the DBS is possible for those DBSs that have interactive language support. The DBS with its language interpreter may execute as a database execution environment. This situation is depicted in Figure III.5.

IV. Types of interfaces in a virtual machine based DSS

We have identified the components in a virtual machine based DSS. We shall assume that each component operates in a separate virtual machine. In this section, we discuss four major functional types of interfaces.

(1) Execution environment to execution environment

An user working in one execution environment may want to run a tool in another execution environment. The tool is initiated in the foreign execution environment but the user carries on the dialog with the tool in the working execution environment. For example, Figure IV.1 depicts the situation where an user in a BASIC language execution environment invokes an APL function in the APL execution environment, BASIC and APL are in different virtual machines.

(2) Execution environment to database system

An user in one execution environment issues data access requests to a database system in another virtual machine. The data is sent to the working execution environment for further analysis. For example, an APL user on one virtual machine interacts with a SEQUEL (Chamberlin and Boyce 74) database system on another virtual machine (Figure IV.2).

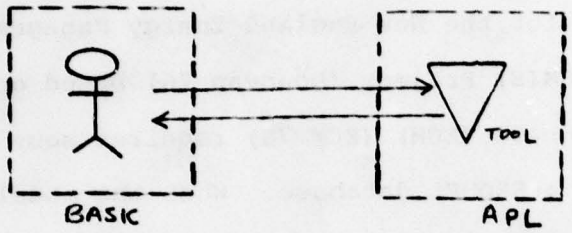


Figure IV. 1 EXECUTION ENVIRONMENT TO EXECUTION ENVIRONMENT INTERFA

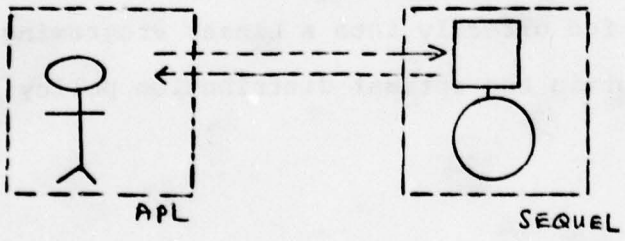


Figure IV. 2 EXECUTION ENVIRONMENT TO DATABASE SYSTEM INTERFACE

(3) Tool to database system

A particular tool running in an execution environment needs some data, it issues the appropriate data requests to a database system in another virtual machine to obtain the data. Data is sent to the tool in the appropriate format. For example, an energy model developed for the New England Energy Management Information System (NEEMIS) Project (Donovan 76) based on the Energy Conservation Manuals (ECM) (ECM 78) requires some data, which may be stored in a SEQUEL database. When the model is invoked, a series of SEQUEL requests are generated to obtain the data and format the data suitable for use by the model (Figure IV.3).

(4) Tool to tool

Two different tools running in different execution environments may be coupled so that output of one tool is fed directly as input to the other tool. For example, output from a macro econometric model, such as prices and expected demand of natural gas, may be fed directly into a Linear Programing planning model to obtain the optimal distribution policy (Figure IV.4).

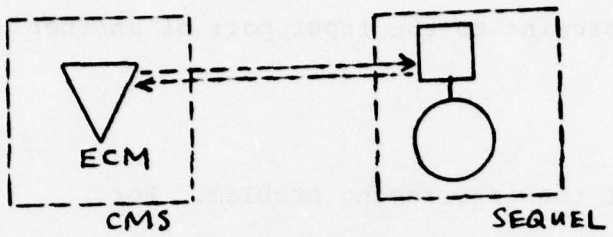


Figure IV.3 TOOL TO DATABASE SYSTEM INTERFACE

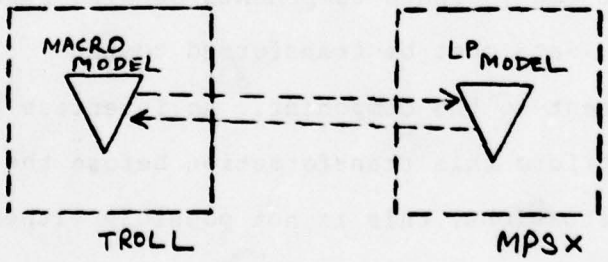


Figure IV.4 TOOL TO TOOL INTERFACE

V. Strategies for portal interfacing of virtual machines

One aspect of interfacing the components in a virtual machine based DSS is that of portal interfacing, i.e. getting data from an output port of a virtual machine to the input port of another virtual machine.

There are other aspects of the interfacing problem. For example, resolving differences in data encoding schemes (e.g. ASCII vs EBCDIC), differences in data structures (e.g. arrays vs hierarchies), and differences in semantics of data (e.g. different units of measure). Thus, once the data is received by the target component via its input port, these differences must be resolved before the data is usable. This task can be performed by an interface residing in either the source or target virtual machine. For certain components, the construction of such an interface may not be feasible. For example, the component may be constructed such that it directly uses the data received via the input port (many canned components behaves like this). For these programs, data must be transformed to the correct form before it is sent to the component. An interface in the source component may perform this transformation before the data is output. In some situations, this is not possible either. We may then use another virtual machine, an interface virtual machine (IVM) to receive data from the source component, perform the necessary transformations to the data, and send the data to the target component (Figure V.1). In this report, we shall

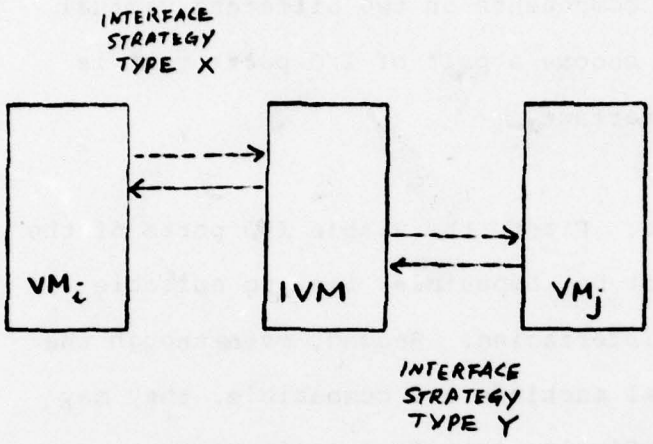


Figure V.1 USE OF IVM AS AN INTERMEDIARY BETWEEN TWO INCOMPATIBLE VMs.

focus on developing strategies to solve the portal interface problem.

V.1 Redirecting I/O ports

An execution environment on a virtual machine limits the set of viable I/O ports to a subset of those supported by the virtual machine. A tool may further limited the set of viable I/O ports it uses. To interface two components on two different virtual machines, we would like to choose a pair of I/O ports that is best for the particular interface.

Two situations may arise. First, the viable I/O ports of the two virtual machines may not be compatible, i.e. no suitable strategy is available for interfacing. Second, even though the I/O ports of the two virtual machines are compatible, they may not be the best for the particular interfacing situation. Several strategies may be used to redirect the data to a different, preferred I/O port. Some of these are listed below.

(1) Change run instructions for the component - it may be possible to simply change the run instructions for the component to redirect output data to another I/O port, e.g. it may be possible to change some Job Control Language (JCL) for a program to redirect the output data designated for a card punch to a tape device.

(2) Intercept I/O calls - it may be possible to intercept I/O calls made by the component and redirect data to a

different I/O port. For example, using the dynamic linkage and search rule mechanisms on MULTICS, one may be able to replace system I/O calls by one's own routines which can be tailored to send data to a prefered I/O port.

(3) Change component - the component may be changed, i.e. replacing all I/O calls in the component to send data to a prefered I/O port. This is often an undesirable strategy because of the reprogramming efforts that entails.

(4) Use of IVM - since the IVM can be designed to use any I/O ports, it can be an effective intermediary between two other virtual machines (Figure V.1).

V.2. Interfacing I/O ports

In the following, we study four major classes of virtual machine portal interfacing strategies. The first type of strategies is direct virtualization of existing mechanisms on real machines. This includes channel-to-channel communications between two real machines, sharing of a disk between two real machines, and telecommunications between two real machines. The second type of strategies is virtualization of existing I/O devices and direct mapping between two virtual devices of two virtual machines. This includes mapping of virtual card reader to virtual card punch, mapping of virtual tapes, and mapping of two similar virtual consoles. The third type of strategies is to extend the capabilities of the VMM to facilitate indirect mapping of virtual I/O devices. This includes mapping of virtual printer

output to a virtual card reader or a virtual console, and the mapping of virtual terminal output to another virtual terminal or to a virtual card reader. The fourth type of strategies is to extend the capabilities of VMM to support inter-virtual machine communications. This includes communications by virtual processor interrupts, specially coupled virtual machines, and shared writable memories. Each of these strategies is discussed in the following sections.

V.2.1 Virtualization of existing mechanisms on real machines

Two real machines may communicate with each other via some mechanism. When these two machines are virtualized, they may still continue to communicate with each other using the same mechanism. Since the two virtualized machines are in the same computer system, some of these communications mechanisms may be simplified. The following are some examples of this strategy.

(1) Channel-to-channel communication

A channel is a processor used to handle I/O, so that CPU and I/O operations can be simultaneous. In some multi-computer systems, where the computer systems are located near one another, high-speed inter-computer communications can be facilitated by using a channel-to-channel adaptor (CTCA) (Figure V.2). The CTCA can be viewed as a real I/O device by either computer. Machine i writes to the CTCA as if it is writing to an I/O device, and

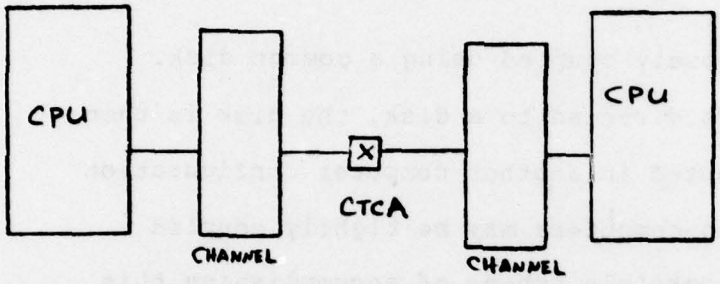


Figure V.2 CTCA

machine j reads from the CTCA, or vice versa. It is possible that one of the machines is virtualized, e.g. a virtual machine may communicate with a real machine using a real CTCA. When both machines are virtualized, the CTCA can be simulated by the VMM, e.g. by using a file instead. This strategy can be very effective for large amount of communications between two virtual machines especially when the two machines have been previously designed to make use of the CTCA.

(2) Shared disk

Two computers may be loosely coupled using a common disk. Output from one computer is directed to a disk, the disk is then physically removed and mounted in another computer configuration as input. Furthermore, two computers may be tightly coupled using a shared disk. One possible scheme of accomplishing this tightly coupled configuration is to make use of a channel switch (Figure V.3). The channel switch allows one CPU to access the disk at one time. For example, machine i may be used to batch input jobs and machine j may be used for normal multiprograming processing. Machine i will put the input jobs in the shared disk using the channel switch, when this has been completed, machine j can read the shared disk to obtain the batch jobs. A synchronization mechanism has to be used so that the two machines will not attempt to use the shared disk at the same time.

On a virtual machine system, the mapping of virtual disk

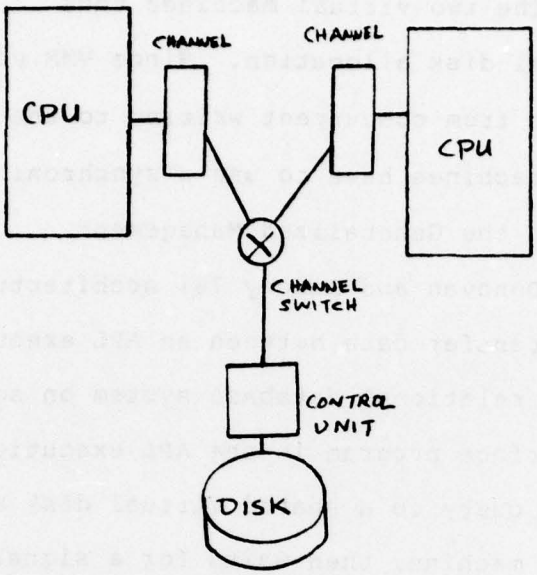


Figure V.3 SHARED DISK

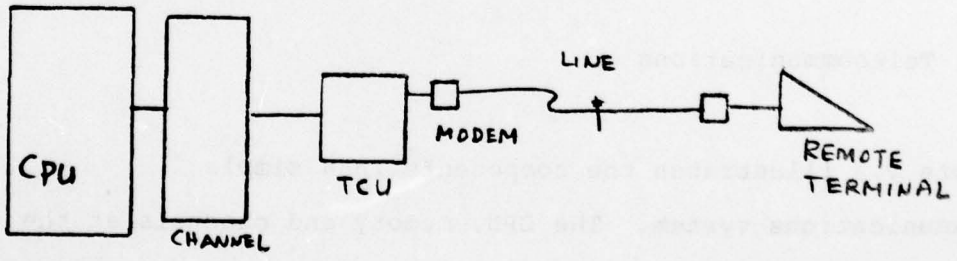


Figure V.4 SIMPLE TELECOMMUNICATION SYSTEM

allocation to real disk is supervised by the VMM. It is possible that a particular virtual disk of one virtual machine be mapped into the same real disk allocation as another virtual disk of another virtual machine. The two virtual machines thus effectively share a physical disk allocation. Since VMM usually does not provide protection from concurrent writing to the shared virtual disk, the virtual machines have to use a synchronization mechanism. For example, in the Generalized Management Information System(GMIS) (Donovan and Jacoby 76) architecture, this strategy is used to transfer data between an APL execution environment and the SEQUEL relational database system on separate virtual machines. An interface program in the APL execution environment writes an user query to a shared virtual disk and signals the SEQUEL virtual machine, then waits for a signal from the SEQUEL machine. An interface on the SEQUEL virtual machine reads the query from the shared virtual disk and passes the query to SEQUEL, the results of the query are written to the shared virtual disk, and the APL virtual machine is notified. The APL interface then reads the data from the virtual disk and performs other processing on the data.

(3) Telecommunications

Figure V.4 illustrates the components in a simple telecommunications system. The CPU, memory and channels at the local computer have been discussed before. The transmission control unit (TCU) is a special I/O device controller that

communicates with remote devices over telecommunication lines. A virtual machine may have a virtual TCU. A virtual TCU is usually a subset of a real TCU, that is, a virtual TCU includes a subset of the communication lines on a real TCU. Using a virtual TCU, a virtual machine may communicate with a remote terminal, or a remote computer. A virtual machine may also communicate with another virtual machine using a virtual TCU and using real communication lines. This may be useful for testing of the communication lines or other telecommunication equipments or communications protocols. Two virtual machines may communicate with each other using simulated communication lines, thus simplifying much of the complexities of communicating over real hardwares. This strategy is useful when a particular tool that has been designed to use a remote terminal is to be interfaced with another component in another virtual machine.

An example of using virtual communications to interface between different execution environments is the Teleprocessing Virtual Machine (TPVM) mechanism developed at the IBM Watson Research Center (Guido and Considine 77). In this system, the TPVM interfaces with real System/7 computers over real telecommunciation lines. An user in the System/7 LABS/7 environment can initiate execution of a tool in a CMS environment executing under a VM/370 computer. The System/7 user can carry on a dialog with the tool directly (Figure V.5).

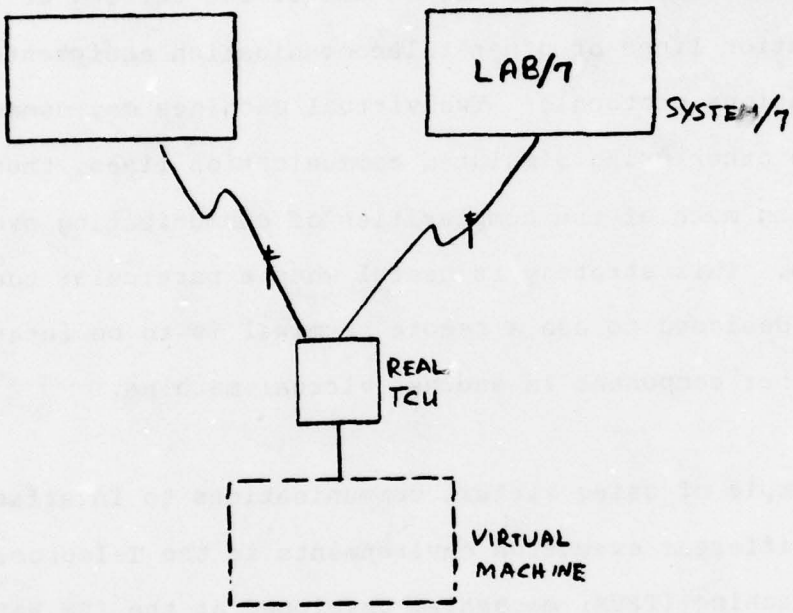
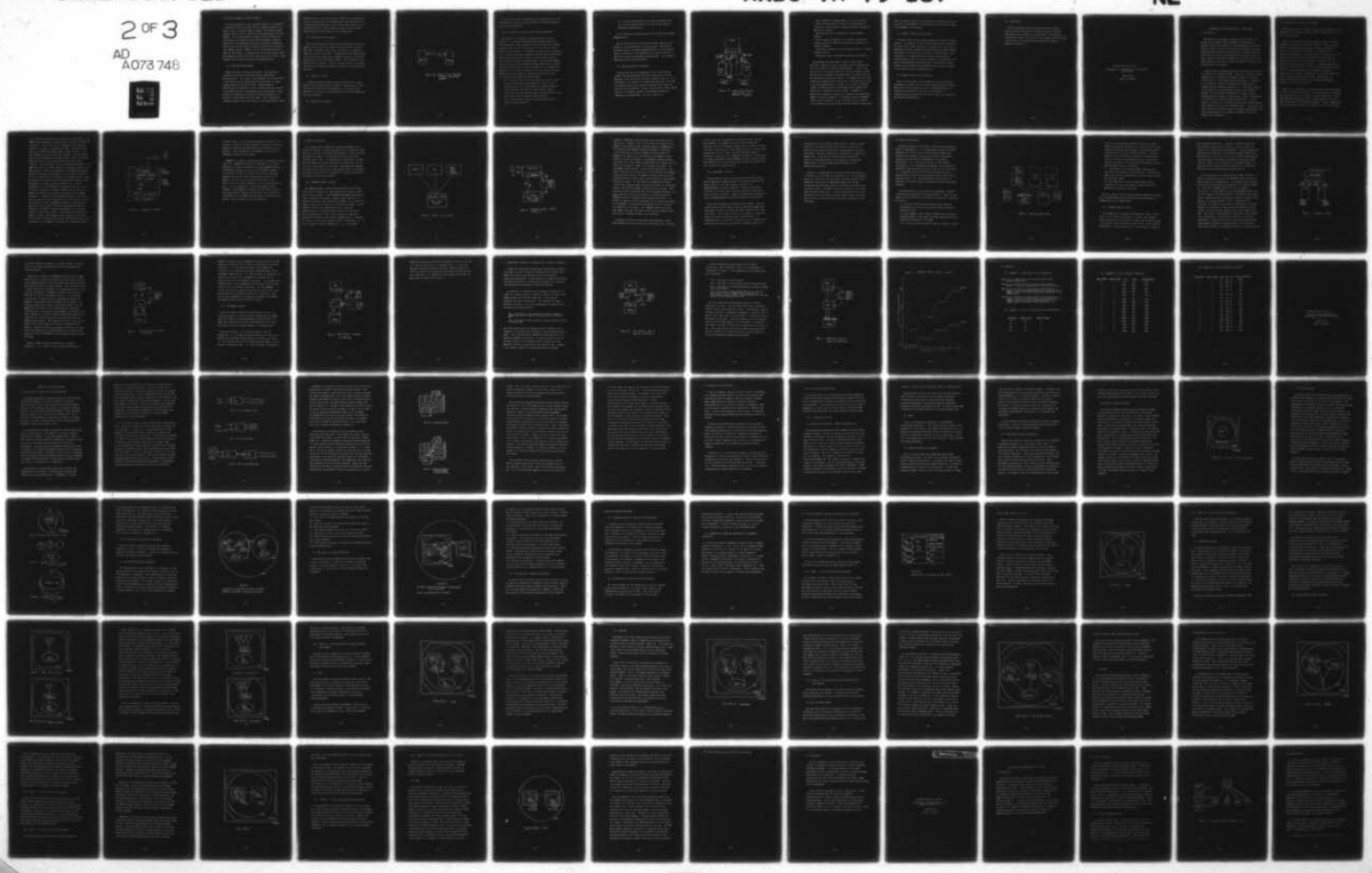
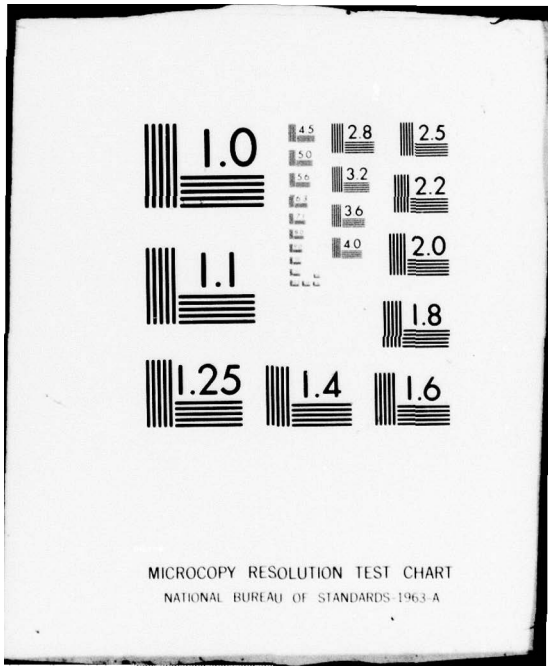


Figure V.5 INTERFACE BETWEEN REAL AND VIRTUAL MACHINE

AD-A073 748 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
USE OF VIRTUAL MACHINES IN THE DEVELOPMENT OF DECISION SUPPORT --ETC(U)
JUL 79 J J DONOVAN, S E MADNICK, C LAM F30602-77-C-0205
UNCLASSIFIED RADC-TR-79-187 NL

2 OF 3
AD A073 748




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

V.2.2 Direct mapping of virtual devices

For certain devices on real computer systems, it is possible to physically carry the output from one computer system to be used as input to another computer system. The loosely coupled computer systems using a shared disk discussed above is one such example. On a virtual computer system, since the devices are virtualized, the transfer of data from one computer system to another may be accomplished by simply moving the corresponding SPOOL files, thus eliminates the human intervention required in a real computer system. However, not all virtual devices can be used for this scheme. It requires that the virtual output device has compatible characteristics as the input virtual device. Some examples of such types of devices are discussed below.

(1) Virtualized tape devices

Tapes are fairly universal I/O devices. Most operating systems can process the few common type of tape device characteristics. Thus output of one computer can be directed to a tape device, and the tape device be carried over to another computer system to be used as input. One strategy of virtualizing tape devices is to use a system disk file, similar to the SPOOL files for virtualizing unit record devices. Data stored on a tape device can be of a variety of formats, e.g. different record length and block sizes. These information, normally recorded on the tape label of a real tape device, can be

recorded with the disk file used for simulating the tape device. To use a virtual tape for interfacing two virtual machines, one virtual machine would write to the virtual tape, after it has finished writing, it would signal the target virtual machine which would then read the data on the virtual tape.

(2) Card punch to card reader

The virtual card punch of one virtual machine can be directly mapped to the virtual card reader of another virtual machine. When data is punched to the virtual card punch, it appears as input the virtual card reader. In Figure V.6, VM(i) punches data to its virtual card punch, which appears directly as data input through the virtual card reader of VM(j). An interrupt is also generated when the data is 'placed' into the virtual card reader, thus the data transfer mechanism also serves as a synchronization mechanism.

(3) Console to console

It may be possible to map the console output of one virtual machine to the console input of another virtual machine, if the two consoles have similar charateristics. Console I/O also generates an interrupt, thus synchronization is also accomplished with data transfer.

(4) Card punch to console

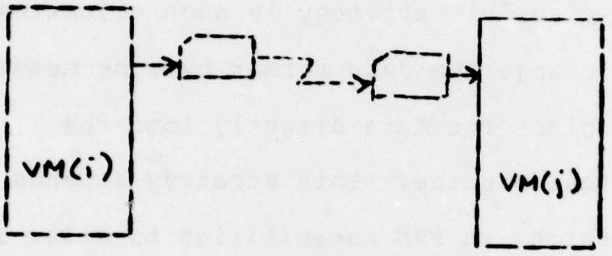


Figure V.6 DIRECT VIRTUAL CARD PUNCH TO VIRTUAL CARD READER MAPPING

If the end of a card is regarded as a carriage return, it may be possible to map the card punch output to the input console of another virtual machine.

V.2.3 Extensions to VMM to support indirect mapping

For devices with dissimilar characteristics, it is not possible to use direct mapping for data transfer between two virtual machines. Some changes to the data format must be made before output from one virtual machine can be used as input to another virtual machine. A plausible strategy in such situations is to intercept the output, change the data format by some means, then use some subroutine to place the data directly into the input device of another virtual machine. This strategy depends on the availability of extensions in VMM capabilities to allow a virtual machine's output to be intercepted, and to provide the subroutines that place the data directly into the input virtual device of a virtual machine. Several situations that such strategy might be useful are listed below.

(1) virtual printer output to virtual card reader input - a printer output record has imbedded in it some control characters and the record length of a printer output is usually greater than that of a card record. These differences must be resolved before the data is sent to the card reader input of another virtual machine.

(2) virtual printer output to virtual console input - similar comment as above.

(3) virtual console output to virtual card reader input

(4) virtual console output to virtual console input of different characteristics.

V.2.4 Extensions of VMM facilities for inter-virtual machine communications

One of the virtues of a virtualized computer system is its ability to isolate different virtual machines. The need for protected sharing of resources among different virtual machines leads to extensions of VMM capabilities to provide facilities that allow inter-virtual machine communications. Three types of VMM extensions are discussed below.

(1) Virtual processor interrupts

As we recall from the discussions of the virtual machine concept, the VMM supervises the entire collection of virtual machines in the way they use the real computer resources. It is not difficult to conceive of a scheme that makes use of the VMM to simulate inter-virtual machine interrupts and to move data from one virtual machine to another virtual machine under program control. For example, consider the following scenario of communications between VM(i) and VM(j) (Figure V.7).

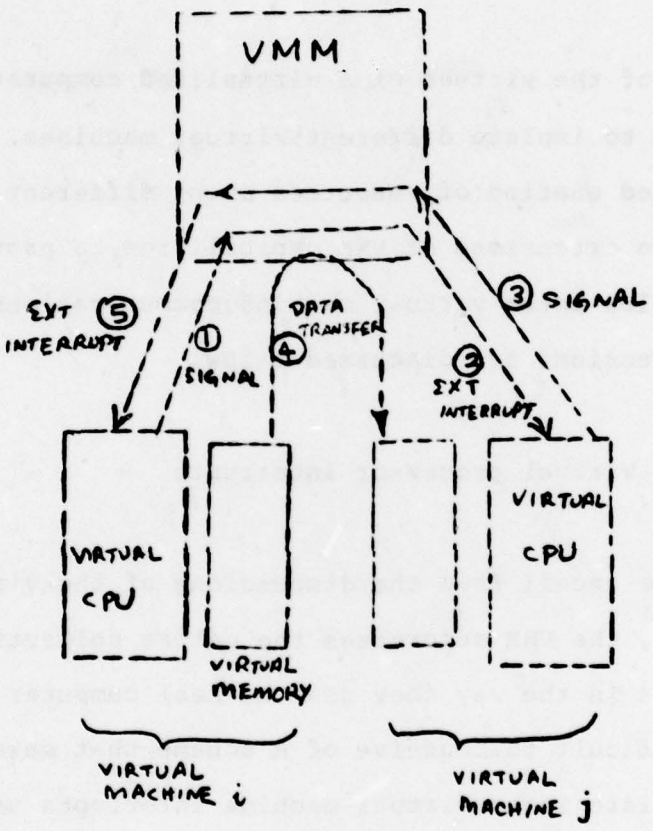


Figure V.7 INTER-VIRTUAL MACHINE COMMUNICATIONS VIA PROCESSOR INTERRUPT

- VM(i) assembles a message (MSG) in its virtual memory.
- VM(i) uses some special convention to signal VMM its intention to send a message, passing to VMM the address of MSG and its length.
- VMM interrupts VM(j) indicating to VM(j) message is pending.
- VM(j) signals the VMM that it is ready to receive the message, passing to VMM the the address of the buffer to hold the message.
- VMM copies the MSG from virtual memory of VM(i) to virtual memory of VM(j).
- VMM signals VM(i) that MSG has been sent and received.

The advantage of this scheme is that it can be quite efficient. VMM may not have to actually copy the message from one virtual memory to another virtual memory. Since it has control of all the system tables for the virtual memories, VMM may be able to accomplish the data movement by simply switching the appropriate system table entries. An example that uses this scheme for transfering data between two virtual machines is used in the GMIS implementation. Briefly, it is used to interface between a TRANSACT execution environment and the SEQUEL base system each on separate virtual machines. TRANSACT is a specialized language environment for accessing and manipulating SEQUEL relational databases. An interface program in TRANSACT takes a query entered by an user and sends it to the SEQUEL virtual machine by using the VM/370 Virtual Machine Communication

Facility routines (IBM 77). Similarly, data results of the query are sent to the TRANSACT virtual machine by an interface program in the SEQUEL virtual machine.

(2) Tightly coupled virtual machines

A virtual machine may be tightly coupled with another virtual machine. For example, certain virtual machines may be authorized to create slave virtual machines. The master and slave virtual machines are tightly coupled such that the master virtual machine can monitor the activities of the slave virtual machines. Some extensions to the VMM capabilities enable the master/slave virtual machine couple to interact closely, e.g. output from the slave machine may be automatically sent to the master virtual machine, and the master virtual machine can sent input directly to the console of the slave machine, etc.

(3) Shared writable virtual memories

VM(i)'s virtual memory can be mapped onto the same physical allocation as another block of virtual memory belonging to VM(j). When either virtual machine writes to the shared memory, the contents are available to the other virtual machine. A synchronization mechanism must be used to ensure the integrity of the shared writable memory.

VI. Conclusions

We have developed concepts that are useful for studying virtual machine based Decision Support Systems and explored strategies that are useful for interfacing virtual machines. A detailed case study of one such decision support system that made use of some of the concepts and strategies will appear in Technical Report No. 2.

Addendum Technical Report 2

Strategies for Interfacing Virtual Machines - A Case Study

Chat-Yu Lam

Stuart E. Madnick

Strategies for interfacing VM's - A Case Study

I Introduction

The Generalized Management Information System (GMIS) is an ad-hoc Decision Support System designed for energy policy analysis (Donovan and Jacoby 75). The characteristics of ad-hoc decisions require that existing tools and databases be quickly brought together to bear on the decisions. To make these tools and databases jointly usable effectively and quickly, there is often a serious interfacing problem that has to be solved. The GMIS approach to solve this interfacing problem is to make use of virtual machine technologies.

Virtual machine technologies enable multiple virtual machines to coexist on one computer system. Each tool or database system, together with its execution environment resides on a separate virtual machine. The virtual machine architecture is exploited in the design of effective interfaces among these tools and database systems (Donovan and Madnick 76). GMIS takes advantage of this approach to allow multiple execution environments on different virtual machines effectively to access a shared database system on another virtual machine. GMIS is implemented on a VM/370 virtual machine system (Donovan 76b). In this report, we focus on the interfaces between two of these execution environments and a relational database system. In particular, we shall study the interfaces between the execution environments and the database system in two stages of the GMIS architecture, called GMIS-I and GMIS-II.

II Description of interfacing components

This section describes the two execution environments, APL and TRANSACT, and the database system, SEQUEL, from the point of view of interface design.

APL is a generalized language interpreter running under the Conversational Monitor System (CMS) (IBM 76). CMS is a single user time-sharing operating system designed specifically to operate on an IBM/370 virtual machine. The APL user can construct APL programs (called APL functions) interactively. The user initiates execution of a saved APL function simply by entering its name, the APL interpreter (listener) will then proceed to process the APL function. All the CMS commands are available to the APL user via the use of the APL shared variable facility (a shared variable can be viewed as an external subroutine call in APL). The APL user can read/write CMS files, also via the shared variable facility. Although APL is a powerful analysis environment, its own database management capabilities are fairly limited.

SEQUEL is an experimental relational database system developed at IBM (Chamberlin and Boyce 74). An improved version of SEQUEL was produced by the joint efforts of the MIT Sloan School and the IBM Cambridge Scienctific Center. SEQUEL operates as a single user system under the CMS operating system. However, SEQUEL does not use the CMS file system, but instead, has its own customized

I/O routines that manage the allocation of its disk spaces. The SEQUEL system consists of several layers of software as depicted in Figure 1. At the lowest level of SEQUEL are the special system I/O routines that manage disk spaces. The next higher level is the Relational Memory (RAM) software (Crick et al 70) that manages the storage of binary relations. The next higher level is the Extended Relation Memory (XRM) (Lorie 74) software which provides an n-ary relation interface to the binary relations. The next higher level is the SEQUEL interpreter. The SEQUEL interpreter translates queries into program calls to the XRM interface. There is an interface to the SEQUEL interpreter, called the Transaction Interface (TI), that system programmers can use to construct user interfaces to the SEQUEL system. Such an interface is called an User Friendly Interface (UFI), and may be designed to interact with the terminal user. The UFI communicates with the SEQUEL TI via program calls and a set of PL/1 global variables to transmit queries from the user to the SEQUEL interpreter and to obtain the results from the SEQUEL interpreter. Typically, an UFI program accepts an user query, puts the query in a PL/1 character string, and calls the TI with the character string. The SEQUEL interpreter translates the query into calls to XRM. When the query processing has been completed, the SEQUEL TI places the description of the query results in some global variable. This description includes the number of tuples, the datatypes of the columns, and so forth. The UFI then interrogates these global variables and calls other entry points in the SEQUEL TI to obtain the individual tuples in

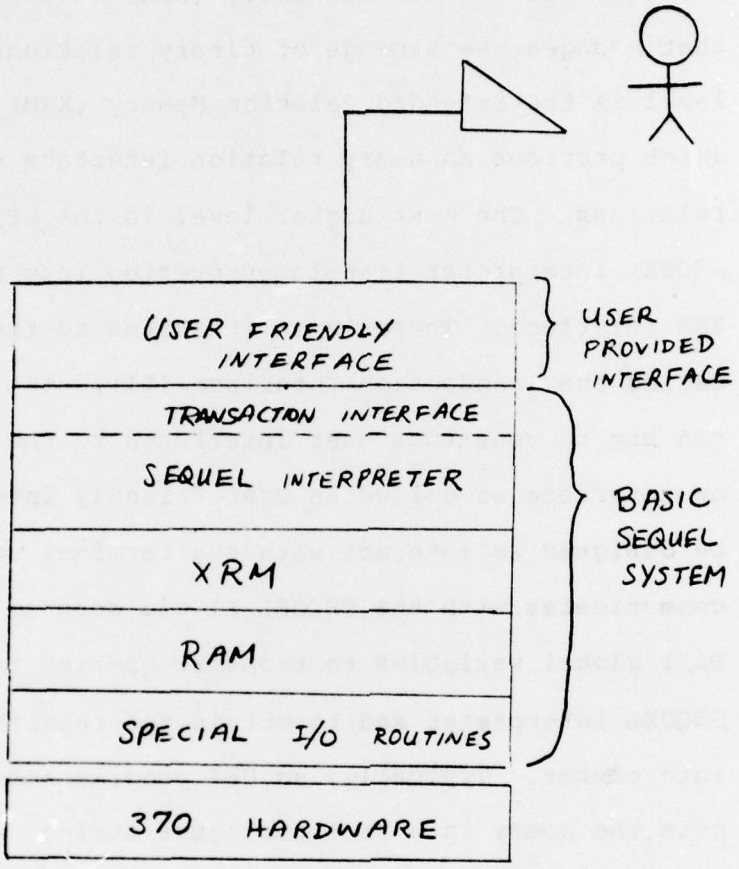


Figure 1 STRUCTURE OF SEQUEL

the query result. The UFI may then display the results on the terminal. A special UFI was developed in GMIS which provides a program interface to programs executing on different virtual machines. This interface will be examined in more detail in subsequent sections of this report.

TRANSACT is a special language execution environment operating under CMS. TRANSACT was specifically designed as a database manipulation language environment to access SEQUEL databases. It was designed to enable an user to use the SEQUEL language interactively without the trouble of writing PL/1 programs. TRANSACT also provides some fairly powerful functions, for example, the user can use the editing functions to aid construction of a group of SEQUEL queries to be executed as an unit. It is also possible to invoke CMS commands within TRANSACT. Thus TRANSACT interfaces with the terminal user to solicitate his/her requests, and processes these requests by interfacing with the SEQUEL database system on another virtual machine. This interface will be studied below.

III GMIS-I architecture

As discussed above, SEQUEL was originally designed to be a single user system. To allow multiple users on different virtual machines to access the database simultaneously, a special UFI was developed for SEQUEL. This interface was called Multi-user interface (MUI). MUI is a collection of programs that interface with the SEQUEL Transaction Interface on the one hand, and with the user virtual machine on the other. MUI stacks the requests from these user virtual machines and process them in a FIFO basis by passing each request to the SEQUEL TI. Figure 2 illustrates the software scheme used on GMIS-I.

III.1 TRANSACT/SEQUEL interface

The TRANSACT/SEQUEL interface mechanism makes use of direct virtual device mapping for communications and a shared virtual disk for data transfer. Since TRANSACT and SEQUEL each operates under a CMS operating system, a shared virtual CMS disk is a convenient means for high volume data transfer (note: recall that SEQUEL itself does not make use of the CMS file system). Direct mapping of virtual card reader/card punch is used for synchronization of the data transfer process (At the time of GMIS-I implementation, there was no extension to VM/370 to provide direct inter-virtual machine communications). Refering to Figure 3, let us trace through a typical transaction. (1) An user in TRANSACT issues a SEQUEL query. (2) An interface

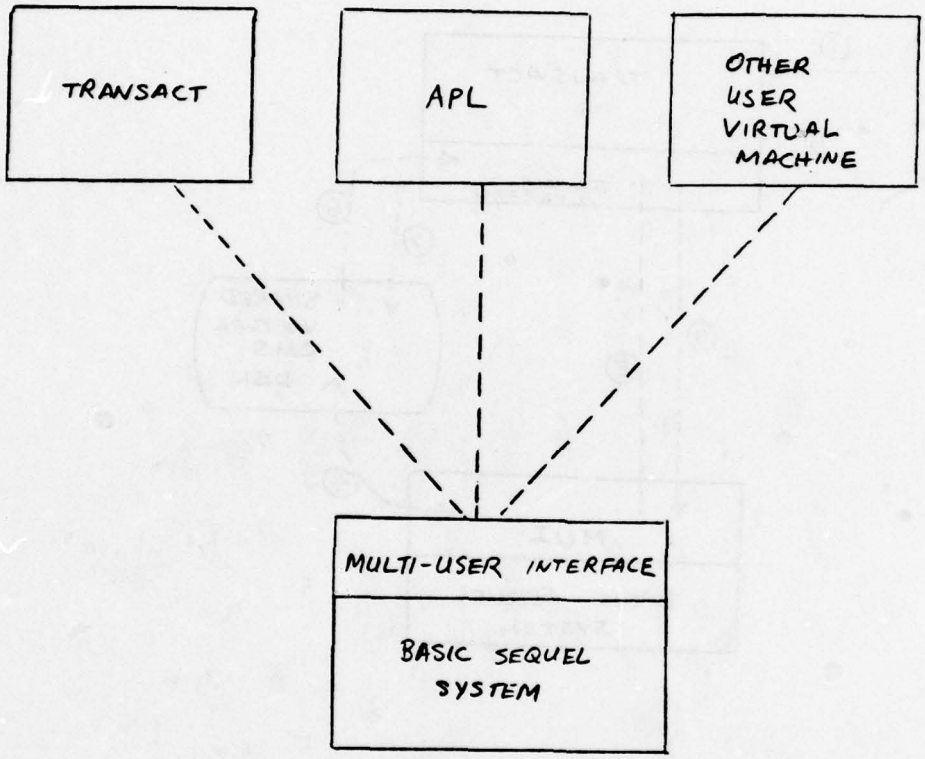


Figure 2 GMIS-I ARCHITECTURE

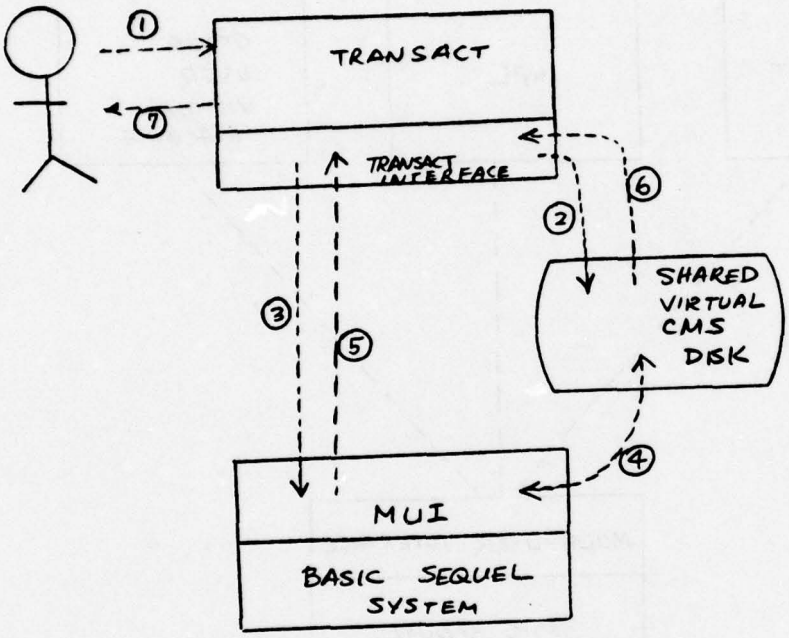


Figure 3 TRANSACT/SEQUEL INTERFACE IN GMIS-I

program in TRANSACT writes this query into the shared virtual disk and (3) punches a card to the virtual card reader of the SEQUEL virtual machine (This interface program in TRANSACT will be refered as the TRANSACT interface hereafter), since this virtual card punch is directly mapped, the card punched by TRANSACT will appear as a virtual card in the SEQUEL virtual machine's virtual card reader. The TRANSACT interface program then waits for a card reader interrupt that will signify that the result of the query is ready in the shared virtual disk. (4) The SEQUEL virtual machine, which had been waiting for a card reader interrupt wakes up when the virtual card from the TRANSACT virtual machine arrives. The SEQUEL interface (MUI) checks the information on the virtual card to find out which virtual machine has sent the query. It then links to the appropriate shared virtual disk, reads the query from the disk (which is in CMS disk format), and calls the SEQUEL interpreter with the query. When the query has been processed, MUI calls the SEQUEL interpreter to obtain the data, then puts the data in a format suitable for TRANSACT and writes the data on the shared virtual disk. (5) Finally it maps its virtual card punch to the virtual card reader of the TRANSACT virtual machine and punches a card to indicate the completion of the query. (6) TRANSACT wakes up from the card reader interrupt and reads the data from the shared virtual disk and (7) displays the data on the terminal.

A few points should be noted about this interface. First, both TRANSACT and the MUI are PL/1 programs under CMS, thus there

is very little data incompatibility between the two. Second, both TRANSACT, the TRANSACT interface, and the Multi-User Interface (MUI) were developed together with the GMIS-I software, thus their interactions are well understood and it is fairly easy to implement the card reader interrupt mechanism simply as additional subroutines to the TRANSACT load module. Third, there is a mutual aggreement between TRANSACT and MUI to use a certain disk address as the shared virtual disk so that the correct disk can be linked for reading/writing.

III.2 APL/SEQUEL interface

The APL/SEQUEL interface mechanism also uses direct virtual device mapping for communications and a shared virtual disk for data transfer for the same reasons as those in the TRANSACT/SEQUEL interface mechanism. A similar interfacing scenario as that described in the last section can be used for this interfacing mechanism. However, several major differences from the TRANSACT/SEQUEL interface mechanism are noted below.

First, APL and SEQUEL use different data formats. APL uses vectors and matrices encoded in special ways while SEQUEL deals with tuples of data with mixed data types (similar to Pl/1 data structures). For the query results to be usable in an APL work space, the data must be structured as APL variables. To accomplish this, the MUI must recognize that it is communicating with an APL environment, and structure the SEQUEL tuples in a

convenient form to be used by an APL function. There is an APL function in the APL work space that reads this data from the shared disk and assigns the values to APL variables in the APL work space. Each APL variable will correspond to a column of the SEQUEL table. Since APL is not particularly suitable for character string manipulations, this process of restructing SEQUEL tuples into APL variables has been found to be a performance bottleneck.

Second, it is undesirable to restructure the APL interpreter to include the card reader interrupt mechanism. This problem was solved by constructing the card reader interrupt handler and its associated routines as an independent module from the APL interpreter and providing a means of initiating this module from an APL work space. The GMIS-I architecture makes use of the APL shared variable capability to invoke the CMS loader which loads and executes the card reader interrupt handler module in another area of virtual storage different from that occupied by the APL interpreter.

IV GMIS-II architecture

Operating experiences with GMIS-I revealed several potential areas for improvement. (1) An experimental version of VM/370 enhanced with a direct inter-virtual machine communication mechanism was available, this provided another interfacing strategy not available for consideration in the GMIS-I architecture. (2) Each time a new user virtual machine environment is to be interfaced with SEQUEL, the MUI has to be modified to add the particular protocol for the new environment. (3) A different user virtual machine interface has to be constructed to interface the user virtual machine with another database system. (4) Some improvements in performance is desirable in the APL interface, especially for high volume data transfers.

A new architecture, called GMIS-II was developed. Figure 4 illustrates the highlights of this architecture. There are five types of virtual machines in the GMIS-II architecture. These are as follows:

(1) The user virtual machines (UVM) provide execution environments that the user can perform his/her analysis or running of tools.

(2) The database virtual machines (DBVM) host the various database systems. They support the data requirements of the execution environments in the UVMs.

(3) An interface virtual machine (IVM) is coupled to an UVM

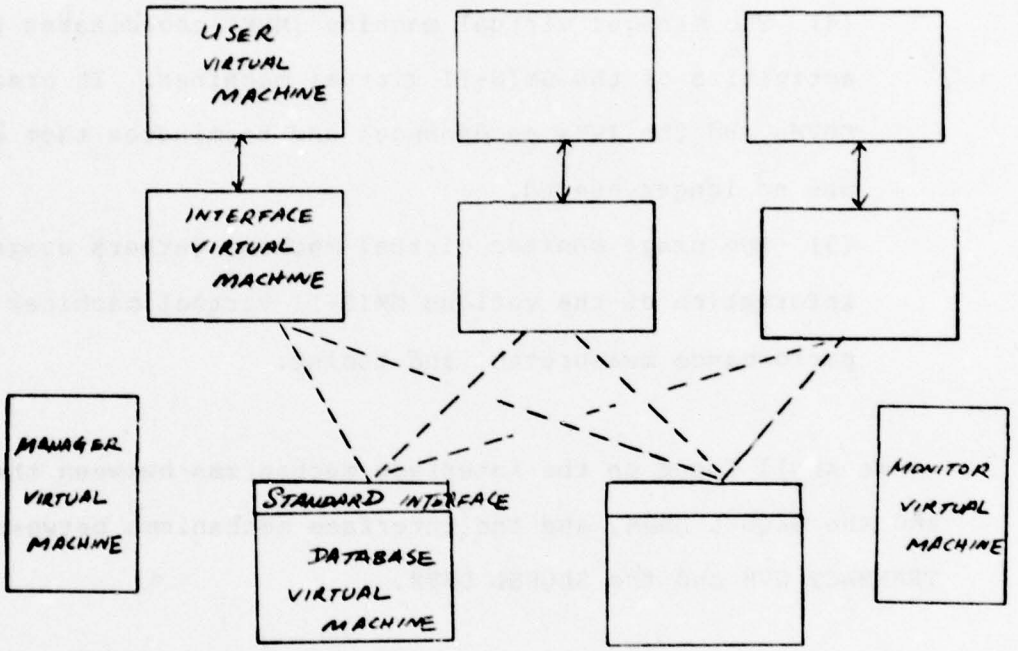


Figure 4 GMIS-II ARCHITECTURE

to act as an intermediary between the UVM and any DBVM. A DBVM has an uniform interface with any IVM in terms of data format and communications protocols. The IVM has standard interfaces with each type of UVM environment. As much as possible of the interfacing processing is concentrated in the IVM interface to make maximum usage of an IVM and to minimize the impact of the interfacing mechanism on existing UVM and DBVM execution environments.

(4) The manager virtual machine (MVM) coordinates the activities of the GMIS-II virtual machines. It creates the DBVMs and the IVMs as demanded and terminates them when they are no longer needed.

(5) The usage monitor virtual machine gathers usage information of the various GMIS-II virtual machines for performance measurement and tuning.

We shall focus on the interface mechanisms between the APL UVM and the SEQUEL DBVM, and the interface mechanisms between the TRANSACT UVM and the SEQUEL DBVM.

IV.1 TRANSACT/SEQUEL interface

A new SEQUEL user interface was developed to replace the MUI so as to use a standard interface to any IVM. This standard interface mechanism is as follows. A query from an IVM is always in the form of a PL/1 variable character string. Data resulting from queries is always structured in a fixed format in blocks of

fixed length storage units. A header is associated with each block describing the data in the block. Data in a block is a packed form of tuples from some relations. The data transfer and synchronization mechanism used between the SEQUEL DBVM and any IVM makes use of the direct inter-virtual machine communications facility called Virtual Machine Communications Facility (VMCF) (IBM 77). VMCF is a new enhancement to the VM/370 monitor. It provides virtual machines with the capability to send/receive with other virtual machines under program control.

A brief description of the VMCF follows (Refer to Figure 5). VMCF is implemented by means of the DIAGNOSE instruction with a special code and a parameter list. The parameter list contains the particular VMCF subfunction requested, e.g. SEND or RECEIVE, etc. Since the DIAGNOSE instruction is a priviledged instruction, a virtual machine issuing the DIAGNOSE instruction will be trapped by the VM/370 monitor. The VM/370 monitor makes use of a special external interrupt to notify the target virtual machine of a pending transfer of data. Along with this interrupt, the target virtual machine also receives a formatted message in a predefined storage location describing the exact nature of the data transfer pending, such as the length of the data. The target virtual machine can then issue a VMCF request with a subfunction to indicate that it wants to receive the pending data. Using this mechanism, two virtual machines with the appropriate external interrupt handler can transfer data under program control. The amount of data that can be moved from

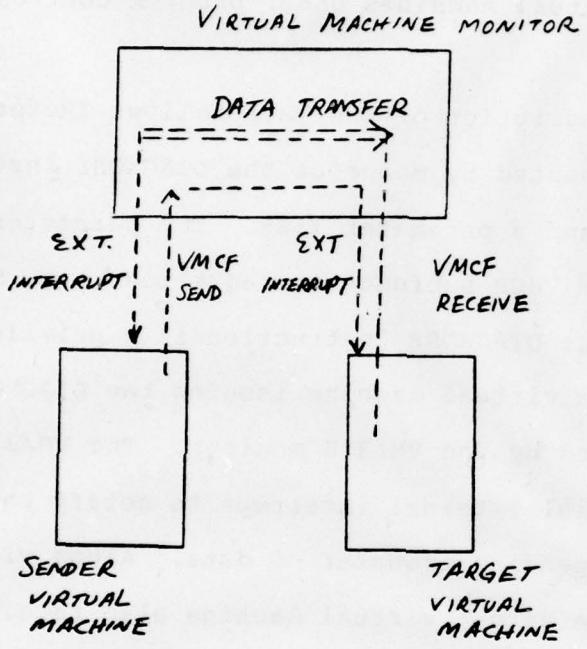


Figure 5 ILLUSTRATION OF VMCF

one virtual machine to another in a single transfer is limited only by the virtual storage sizes of the two communicating virtual machines.

Recall that in GMIS-I, the TRANSACT interface and the SEQUEL MUI make use of a shared virtual disk for data transfer and use direct mapping between virtual card reader and virtual card punch for synchronization of the data transfer. The MUI was also able to detect which user environment it is communicating with, and if this user environment is TRANSACT, the MUI formats the data specially for use by the TRANSACT interface. In the GMIS-II architecture, an IVM is used as an intermediary between the TRANSACT virtual machine and the SEQUEL virtual machine. The interface program that resides on the IVM interfaces with the SEQUEL user interface via the standard interface mechanism described above. To quickly free the SEQUEL virtual machine for other transactions, the IVM interface program also makes use of a temporary disk to quickly unload the data received from the SEQUEL virtual machine. This temporary disk is in fact used as a shared virtual disk between the IVM and the TRANSACT virtual machine. Instead of using direct mapping of virtual card reader and virtual card punch for synchronization of the data transfer process, the IVM and TRANSACT virtual machine use VMCF to signal each other.

Refer to Figure 6 for an illustration of a typical transaction. (1) A query is sent to IVM using VMCF from the

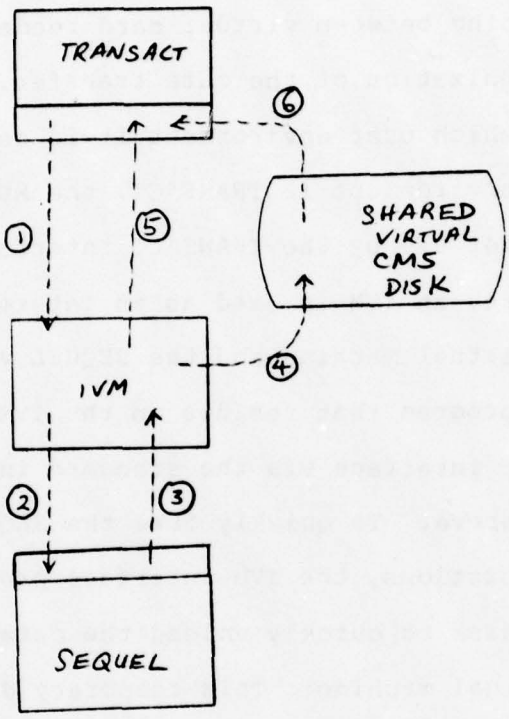


Figure 6 TRANSACT/SEQUEL INTERFACE IN GMIS-II

TRANSACT interface, then the TRANSACT interface waits for an VMCF message from IVM which will indicate the completion of the query processing. (2) IVM sends this query to the SEQUEL user interface via the standard interfacing mechanism common to any IVM-SEQUEL communications. (3) When the query is completed, IVM receives the results from the SEQUEL user interface and (4) stores the data into the shared virtual disk. (5) The IVM sends a VMFC message to notify TRANSACT virtual machine that the query results are in the shared virtual disk. (6) The TRANSACT virtual machine then reads the data from the shared virtual disk and may perform any subseqent processing with the data. Note that no data reformating is necessary since SEQUEL and TRANSACT use compatible data formats (PL/1 data format).

IV.2 APL/SEQUEL interface

The APL to SEQUEL interface also makes use of an IVM. An interface program in the IVM is used to communicate with SEQUEL using the standard interfacing mechanism between any IVM and SEQUEL and communicates with APL using shared disk and virtual card punch/virtual card reader mappings, as discussed above.

Refering to Figure 7, let us follow through a typical transaction. (1) The APL interface writes an user query to the shared disk and signals IVM by punching a virtual card to IVM. (2) IVM reads the query from the shared disk and sends it to the uniform interface via VMCF. (3) After the query is processed by

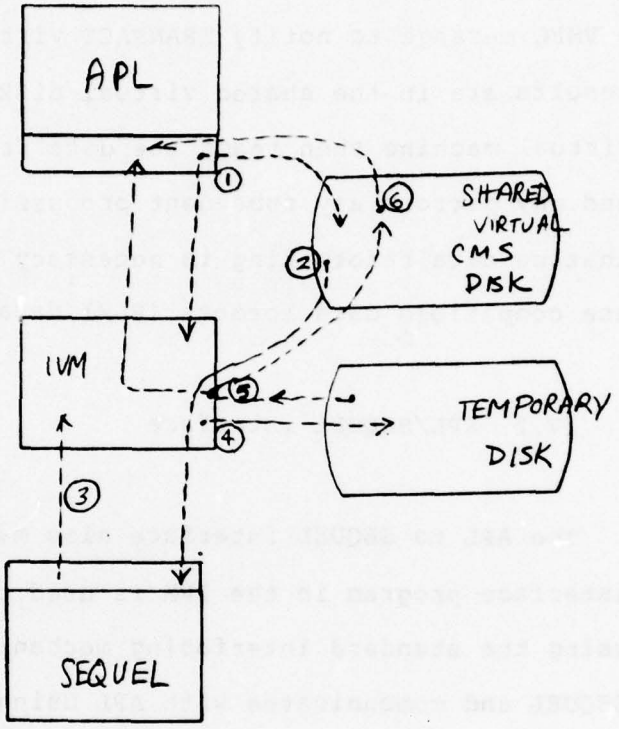


Figure 7 APL/SEQUEL INTERFACE IN GMIS-II

SEQUEL, the results are sent by the uniform interface to the IVM. (4) IVM writes the data to a temporary disk. (5) IVM then reformats the data to APL file variable internal formats and writes the data to the shared disk and signals the APL interface by a punch card. (6) The APL interface wakes up and reads the data from the shared virtual disk.

V. Experimental analysis of overhead costs in GMIS-I and GMIS-II

A significant performance improvement was obtained in GMIS-II over GMIS-I by using the IVM to restructure data received from SEQUEL and using a more efficient inter-virtual machine communication mechanism. In this section, we examine briefly two experiments performed to estimate the overhead costs incurred by the communications mechanisms in GMIS-I and GMIS-II.

An experiment was performed with the GMIS-I architecture, to study the overhead incurred in sending a request for data to the SEQUEL machine from the APL machine and in receiving the requested data back (Donovan 76a). The configuration used is depicted in Figure 8. Part of the experiment was measuring the overhead costs associated with :

A(1) = time spent in APL.interface (passing commands to SEQUEL and converting data from SEQUEL to APL variables); and

B(1) = time spent in MUI (linking to shared disk and writing to shared disk)

The total overhead cost was hypothesized as a function of the complexity of the query and the amount of data resulting from the query. Four levels of query complexities corresponding to the queries in Appendix A were used. To vary the amount of data in the query results, four different sizes of tables were used. Appendix B shows the sizes of the four relations used. Appendix C is a table of some of the timing measurements obtained.

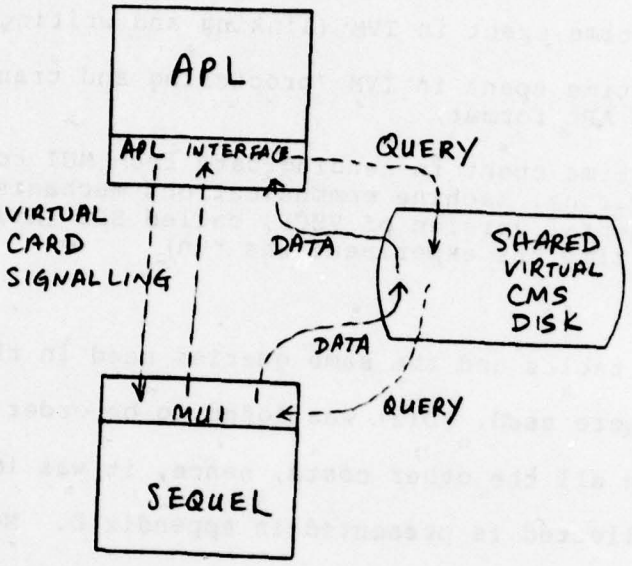


Figure 8 CONFIGRATION USED IN GMIS-I EXPERIMENT

A similar experiment was performed with the GMIS-II architecture. The configuration used in that experiment is illustrated in Figure 9. Four components of the overhead costs were identified:

A(2) = time spent in APL interface

B(2) = time spent in IVM (linking and writing shared disk)

C(2) = time spent in IVM (processing and transforming SEQUEL data to APL format)

D(2) = time spent in sending data from MUI to IVM using inter-virtual machine communications mechanism (an experimental version of VMCF, called SPY (Hsieh 74) was used at the time the experiment was run)

The same tables and the same queries used in the GMIS-I experiment were used. D(2) was found to be order of magnitude smaller than all the other costs, hence, it was ignored. Some of the data collected is presented in Appendix D. Note that B(2) is equal to B(1) since this is the time spent in linking and writing the shared virtual disk. Figure 10 displays the total overhead costs in the two experiments on the same graph. It indicates that GMIS-II was able to reduce the overhead costs by a factor of two. This comes from moving the data reformat processing from the APL interface to the IVM, and using a much more efficient inter-virtual machine communication mechanism.

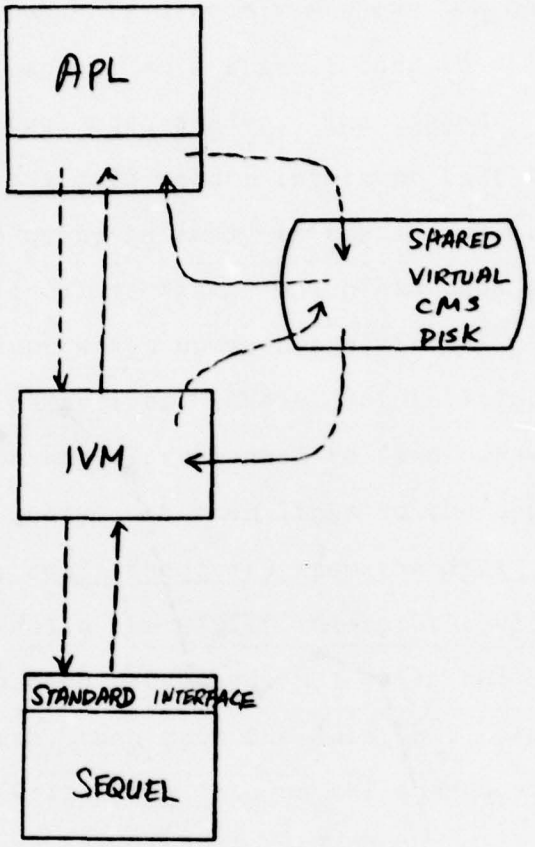
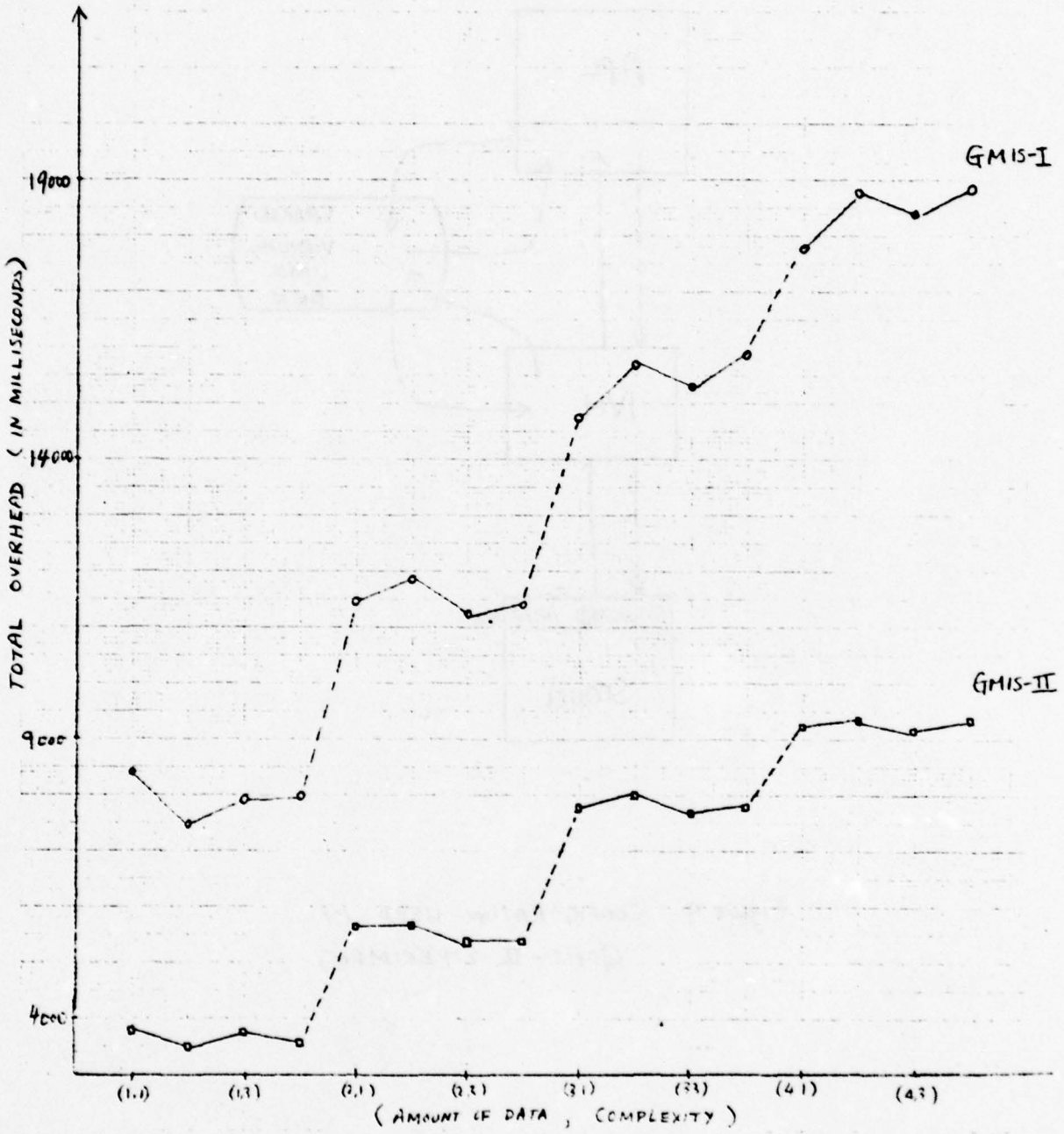


Figure 9 CONFIGURATION USED IN GMIS-II EXPERIMENT

Figure 10. COMPARISON OF GMS-I VS GMIS-II OVERHEAD



VI Apendices

VI.1 Appendix A - Queries used in the experiments

level-1 -- a simple retrieval of all data from a table
(SELECT * FROM table;)

level-2 -- a retrieval of all data meeting one condition
(SELECT * FROM table WHERE STATE IN (CT,VT,ME,NH,RI,MA);)

level-3 -- a retrieval of all data meeting two conditions
(SELECT * FROM table WHERE STATE IN (CT,VT,ME,NH,RI,MA) OR
STATE IN SELECT STATE FROM INCOME WHERE TWOTOTHREE = Ø OR
STATE = MA;;)

level-4 -- a retrieval of all data meeting three conditions
(SELECT * FROM table WHERE STATE IN (CT,VT,ME,NH,RI,MA) OR
STATE IN SELECT STATE FROM INCOME WHERE COUNTY IN
SELECT COUNTY FROM TERMINAL WHERE PLACEMENT = Ø;;;)

VI.2 Appendix B - Sizes of tables used in the experiments

Table Name	Number of Rows	Number of Columns
R1	457	2
R2	457	4
R3	457	6
R4	457	8

VI.3 Appendix C - Data from GMIS-I experiment

Amt. of Data	Compl. of Query	A(1)	B(1)	Total=A(1)+B(1)
1	1	3957	4505	8462
1	2	4381	3162	7543
1	3	4711	3399	8110
1	4	4695	3332	8027
2	1	6430	5128	11558
2	2	6839	5077	11916
2	3	6402	4939	11341
2	4	6509	4973	11482
3	1	7838	7001	14839
3	2	8656	7152	15808
3	3	8563	6926	15489
3	4	8899	7019	15918
4	1	9805	8064	17869
4	2	10769	8188	18957
4	3	10491	8016	18507
4	4	10849	8118	18967

IV.4 Appendix D - Data from GMIS-II experiment

Amt. of Data	Compl. of Query	A(2)	B(2)	C(2)	Total=A(2)+B(2)+C(2)
1	1	304	4505	40	3849
1	2	301	3162	40	3505
1	3	350	3399	41	3790
1	4	306	3332	42	3680
2	1	518	5128	41	5687
2	2	516	5077	40	5633
2	3	515	4939	40	5494
2	4	527	4973	41	5541
3	1	810	7001	41	7852
3	2	830	7152	42	8024
3	3	827	6926	41	7794
3	4	830	7019	42	7891
4	1	1165	8064	41	9270
4	2	1170	8118	42	9330
4	3	1163	8016	42	9221
4	4	1178	8118	43	9339

Addendum Technical Report 3

Composite Information Systems - A New Concept in Information Systems

Chat-Yu Lam

Stuart E. Madnick

Composite Information Systems

I Motivations for Composite Information Systems

As problems confronting organizations and our society become increasingly complex, it has been necessary to find more effective tools to aid in policy analysis and decision-making. This increased complexity and urgency is the result of factors such as : (1) the scarcity and unreliability of many essential natural resources, most notably energy (Donovan 76a), and (2) the subtle interdependence of many decisions where an 'optimal' decision to resolve one problem may in fact, precipitate worse problems in other areas (Forrester 75).

At the same time, the number of powerful problem-solving tools have increased rapidly. For example, at one time, solving a large linear program was a major programming effort; today, a large number of general purpose LP packages are available so that the user can concentrate on problem-solving rather than programming. Large and powerful econometric modelling facilities and models are available and frequently used for policy analysis; statistical and analytical packages are widely used in Management Science; database systems have become essential tools in handling the complex data requirements of organizations; and many other major tools are available and in common use.

Unfortunately, these problem-solving tools are often not compatible with one another. The developers of these tools seldom work in close coordination. Furthermore, it is not

possible to anticipate and to allow for the different ways the tools will be used in the future. Thus we are faced with a difficult situation. On the one hand, most non-trivial problems require powerful tools to aid the problem solution, on the other hand, existing tools are often incompatible with one another, and it is highly unlikely that an unified development of these tools will emerge in the future. Efforts to integrate these separate tools into a single high-level process have been difficult to accomplish. The overhead, inconvenience, inefficiencies, and complexities associated with repeated crossing of system boundaries have been prohibitive.

Let us illustrate some of these concepts using a hypothetical case. The Ozo Gas Company is a major distributor of natural gas in the Mid-West. It acts as an intermediary between the gas suppliers and the consumer and its major profit comes from efficient distribution of natural gas. The company has developed a LP model of the gas distribution process and has been using it as basis for policy making. Basically the LP model takes projected supply and demand information and generates an optimal distribution schedule as well as the projected profit associated with the schedule. This is illustrated in Figure I.1. All was well until the 1973 Oil Embargo. Since then government has been considering new natural gas policies. (There has been a price ceiling on domestic natural gas and government is considering alternative policies to encourage natural gas production)

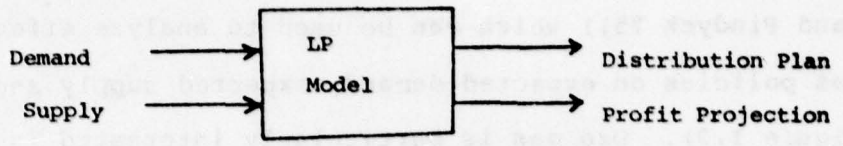


Figure I.1 Ozo Gas Company LP Model

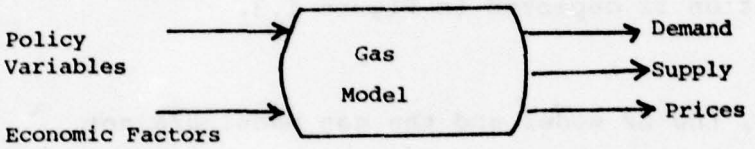


Figure I.2 National Gas Model

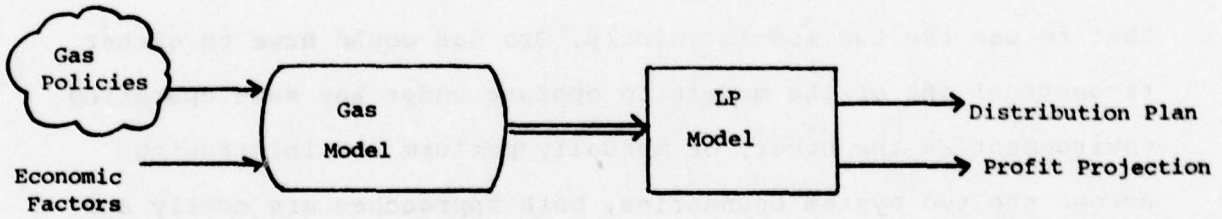


Figure I.3 What Ozo Gas Company wants

Management is anxious to find out how the various new policies would affect its profit from the distribution process. They learned that experts in Cambridge have built a powerful national gas supply and demand model (e.g. the MacAvoy-Pindyck gas model (MacAvoy and Pindyck 75)) which can be used to analyze effects of various gas policies on expected demand, expected supply and prices (Figure I.2). Ozo gas is particularly interested in using the output from the national gas model as input to its LP model, so that by varying the parameters input to the gas model, corresponding to the various new gas policies, the effects of these policies on the gas distribution process, e.g. optimal distribution schedules and profit levels, can be determined. This ideal situation is depicted in Figure I.3.

Unfortunately, the LP model and the gas model are not compatible with each other. The LP model was coded in APL and uses OS/360 file system. The gas model is expressed in TROLL (NBER 75b), a powerful econometric modelling subsystem designed to operate under the VM/370 operating system using its own file system. This situation is illustrated in Figure I.4. It seems that to use the two models jointly, Ozo Gas would have to either reconstruct one of the models to operate under the same operating environment as the other, or manually perform the interfacing across the two system boundaries, both approaches are costly and time consuming for such one-time decisions. What would seem desirable to Ozo Gas is a facility that can incorporate these two models easily and rapidly so that the two models can be jointly

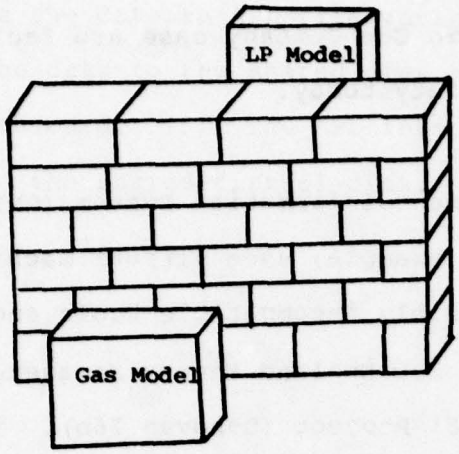


Figure I.4 Incompatible Models

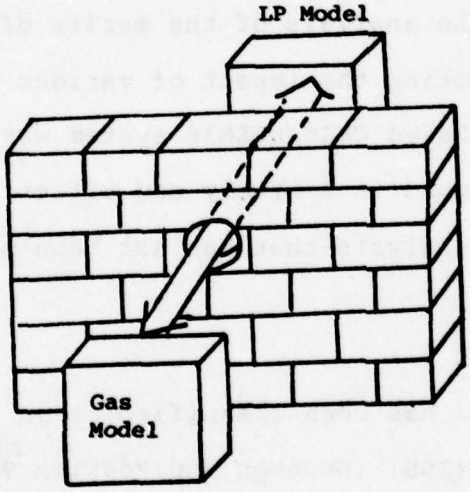


Figure I.5 Result of effective interface between incompatible models

usable. That is, a facility that breaks the wall between the two models with minimal changes to either model (Figure I.5). Similar problems as the Ozo Gas Company case are facing many organizations and our society today.

The Generalized Management Information System (GMIS) (Donovan and Jacoby 75) at MIT for example, uses virtual machines to host many powerful, though possibly incompatible tools and database systems in support of the New England Energy Management Information System (NEEMIS) Project (Donovan 76b). The purpose of the project was to develop an information system for the New England Regional Commission to assist policy makers in managing the problems brought on by the energy crisis. Better information was needed to analyze methods to conserve fuel and to assess the impact of tariffs and prices on different industrial sectors and states within the region. Support was also required for many other studies, such as the analysis of the merits of refineries in the region, and monitoring the impact of various conservation measures. The computer-based GMIS/NEEMIS system was developed to allow researchers to respond in a timely and effective way, to demands for information analysis that has not been previously anticipated.

The GMIS/NEEMIS system has been classified as an ad-hoc Decision Support System (DSS) (Donovan and Madnick 76b) to differentiate it from another type of DSS called institutional DSS. An institutional DSS supports a recurring decision of a

particular type, for example, the decision of a portfolio manager to buy or sell bonds (Gerrity 71). An ad-hoc DSS must address a wider range of decision types. For example, decisions on the allocation of energy resources is a decision type within a logical group of energy related decisions. A major characteristics of an ad-hoc DSS is that the decision and the data needs for its analysis are not known long in advance, there is great time pressure for information, and the perception of the decision as well as the needs for its analysis often change as more information is made available to the decision-maker. Thus there is a need for an approach that allows existing, often incompatible, problem-solving tools to be rapdily integrated at low fixed inital cost to solve problems of great complexity. One approach to meet these needs is to provide a facility that can host these tools and incorporate new tools easily and quickly. We call such an facility and its tools a Composite Information System (CIS) and this approach of information system development the CIS approach. In this report, we develop concepts useful for studying CIS and examine several existing system development efforts that made use of the CIS approach in varying degrees.

II Concepts for studying CIS

A CIS is a computer facility to host a variety of components (e.g. database systems, models, and tools) that may have been developed by different people, often for different purposes. These components often operate in different operating environments and use very different types of databases. Thus, various types of incompatibilities among these components may arise. To make these components jointly usable, the problems associated with crossing system and data boundaries have to be solved.

An effective CIS provides the capabilities to facilitate component integration. Various strategies can be used to solve the problems associated with crossing system and data boundaries. Hence, an effective CIS provides a computing environment to facilitate the joint usage of an assortment of components so that the user of these components does not have to be concerned with computer problems, but to concentrate on solving his(her) problems.

In Section II.1, we clarify some concepts regarding types of components in a CIS, and discuss several types of data and system incompatibilities among these components. These characteristics of a CIS constitute a structural model of the CIS. In Section II.2, several strategies that may be used in a CIS for resolving incompatibilities among components are discussed.

II.1 A Structural Model of CIS

A structural model of a CIS describes the various components in the CIS with their associated operating environments, and the various incompatibilites among these components across system and data boundaries. In this section, we shall discuss four types of components in a CIS, then the various types of system and data boundaries that may exist among these components are discussed.

II.1.1 Components in a CIS

(1) Execution environments - Operating Systems (OS)

An operating system supervisor (OSS) manages the resources in a computer system. The OSS provides macros and subroutines that a program can invoke to utilize the computer resources, e.g. to read a file on a disk. The OSS also provides special utility programs that an user can invoke to perform various maintenance functions, e.g. allocate space for a file, copy a file, etc. An user communicates his/her requests for service to the OSS via a control language. There is a component of the OSS that listens for user requests. For example, in the case of a non-multiprogrammed OSS, the listener may pick up a request from a user to run program X. The OSS initiates program X and passes control of the central processor unit (CPU) to program X. When program X finishes its processing, control of the CPU is returned to OSS and the listener is reinitiated to wait for more user

requests. We often refer to this OSS loop as a listening loop.

From the point of view of the user within the OSS listening loop, the OSS, its utilities, and the computer resources collectively provide him/her an execution environment to construct new programs, run existing programs and perform maintenance functions provided by the OSS. We refer to this type of execution environment loosely as an operating system (OS).

(2) Tools

A tool is a program that represents a well-defined, closed-ended computation. A tool usually receives control from a supervisor of the execution environment that the tool is in. It performs its computation and may carry on a dialog with the user during its computation, typically to obtain some input. At the end of the computation, the tool returns control to the execution environment supervisor.

(3) Special execution environments

There is another class of programs that behave quite differently from tools. We refer to this class of programs as special execution environments. A program of this type receives control from the execution environment supervisor, but does not immediately return control to the supervisor. In fact, the program proceeds to set up its own listening loop and in effect

traps the user in another execution environment. In general, the user communicates with such a special execution environment via a command language (e.g., an APL interpreter). The user is able to construct new tools, run existing tools and may be able to perform some maintenance functions within the special execution environment. Although eventually control is returned to the execution environment supervisor, by explicit user request, the computation represented by this program is ill-defined, and open-ended.

We have identified two major subclassses of special execution environments. These are the generalized language interpreters and the special purpose language processors.

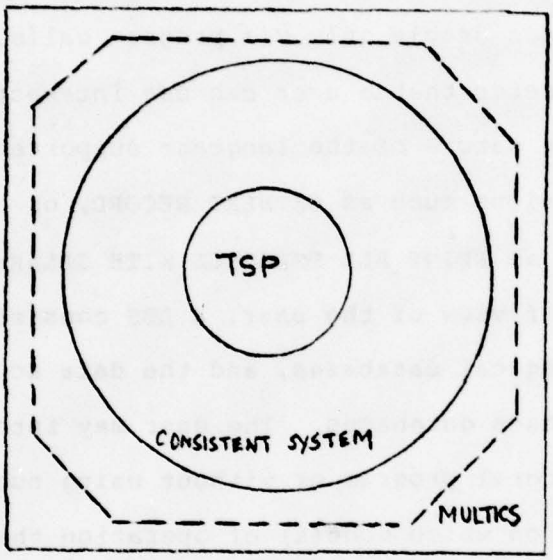
(a) Generalized language interpreters

Generalized language interpreters include all the interpretive programming languages, e.g. APL, BASIC, and so forth. A generalized language interpreter usually is directly under an operating system. The listener of the execution environment is the language interpreter. Under this execution environment, the user is able to construct new tools interactively, run existing tools, and to perform some maintenance functions. These maintenance functions are often a subset of those available from the host operating system. There is considerable variations in the scope of these functions across different host operating systems. For example, under the Cambridge Monitor System (CMS)

(IBM 76), an APL user can invoke most of the CMS commands via the APL shared-variable facility, however, under OS/VS1, the APL user has very limited access to OS/VS1 system maintenance functions.

(b) Special language processors

Specialized language systems have been developed to reduce user programming efforts and to better suit the particular application interests of the users. For example, special systems have been deisgned for construction of simulation models, for construction of econometric models and for data analysis, and so forth. Many of these systems behave like a special execution enviornment we discussed above. Furthermore, due to the special application orientation of most of these systems, they often have their own special file systems particularly suitable for their applications. Thus, the maintenance functions available to the user in such a special execution environment may be quite different from those available to the user if he/she were in the host execution environment. Notice also that an execution environment may reside on another execution environment which may reside on yet another execution environment, and so forth. For example, Figure II.1 illustrates that a modified version of the Time Series Processor (TSP) runs as a special language processor under the Consistent System (Cambridge Project 74c), which is a special execution environment for social science research that runs under the MULTICS operating system on a Honeywell 6180 computer.



HIS/6180

Figure II.1 ILLUSTRATION OF EXECUTION ENVIRONMENTS

(4) Database systems

A database system (DBS) consists of two basic components: the databases and a program that manages these databases and provides data access interface(s) for the user of the databases. The databases may be constructed using very complex data structuring techniques, e.g. indexing, chaining, etc, or they may be constructed using very simple data structuring techniques, e.g. sequential file organizations. The data access interface, also referred to as the data manipulation language (DML) could be a set of subroutines usable only via program calls, or it may be a language interpreter that a user can use interactively. In either case, the nature of the language supported may be either low level operations such as GETNEXT RECORD, or high-level operations such as PRINT ALL EMPLOYEE WITH SALARY OVER 35000. From the point of view of the user, a DBS consists of a collection of logical databases, and the data access language(s) to manipulate thses databases. The user may interact with the DBS via a procedural program or without using such a program, or both, depending on which mode(s) of operation that the DBS supports.

In some cases a DBS may be used as a tool. For example, a specific application program and the DBS software may be linked into one load module and executed as a giant program, like an ordinary tool. This situation is depicted in Figure II.2. Since a DBS is a very expensive resource and is often used to support

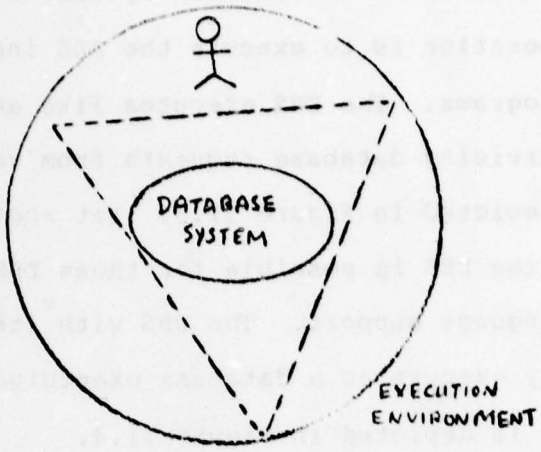


Figure II.2 DATA BASE SYSTEM USED AS A TOOL

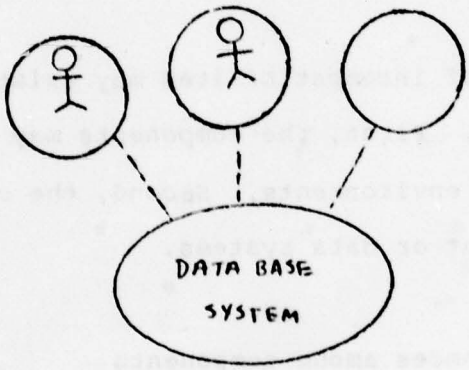


Figure II.3 DATA BASE SYSTEM SERVICING MULTIPLE EXECUTION ENVIRONMENTS

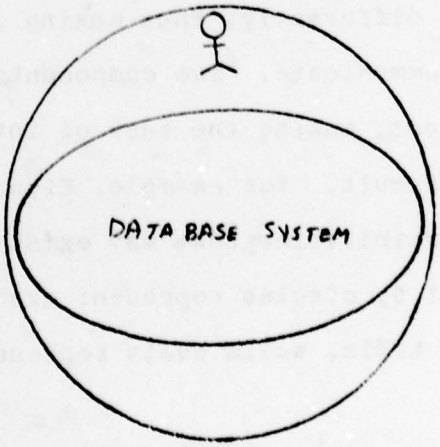


Figure II.4 DATA BASE SYSTEM USED AS AN EXECUTION ENVIRONMENT

all the data needs of an information system, a frequently used mode of DBS operation is to execute the DBS independent of other application programs. The DBS executes like an execution environment servicing database requests from various users. This situation is depicted in Figure II.3. Yet another mode of operation for the DBS is possible for those DBSs that have interactive language support. The DBS with its language interpreter may execute as a database exectuion environment. This situation is depicted in Figure II.4.

II.1.2 Incompatibilites among CIS components

Two basic types of incompatibilites may exist between components in a CIS. First, the components may operate on different operating environments. Second, the components may use different data format or data systems.

(1) Data differences among components

Two components may use the same type of file system, but each may encode the data differently, thus making it difficult for the two components to communicate. Two components may use entirely different file systems, making the task of integrating these two components more difficult. For example, Figure II.5 illustrates various data incompatibilities that may exist among components in a CIS. In Figure II.5, circles represent execution environments, triangles represent tools, solid ovals represent data systems,

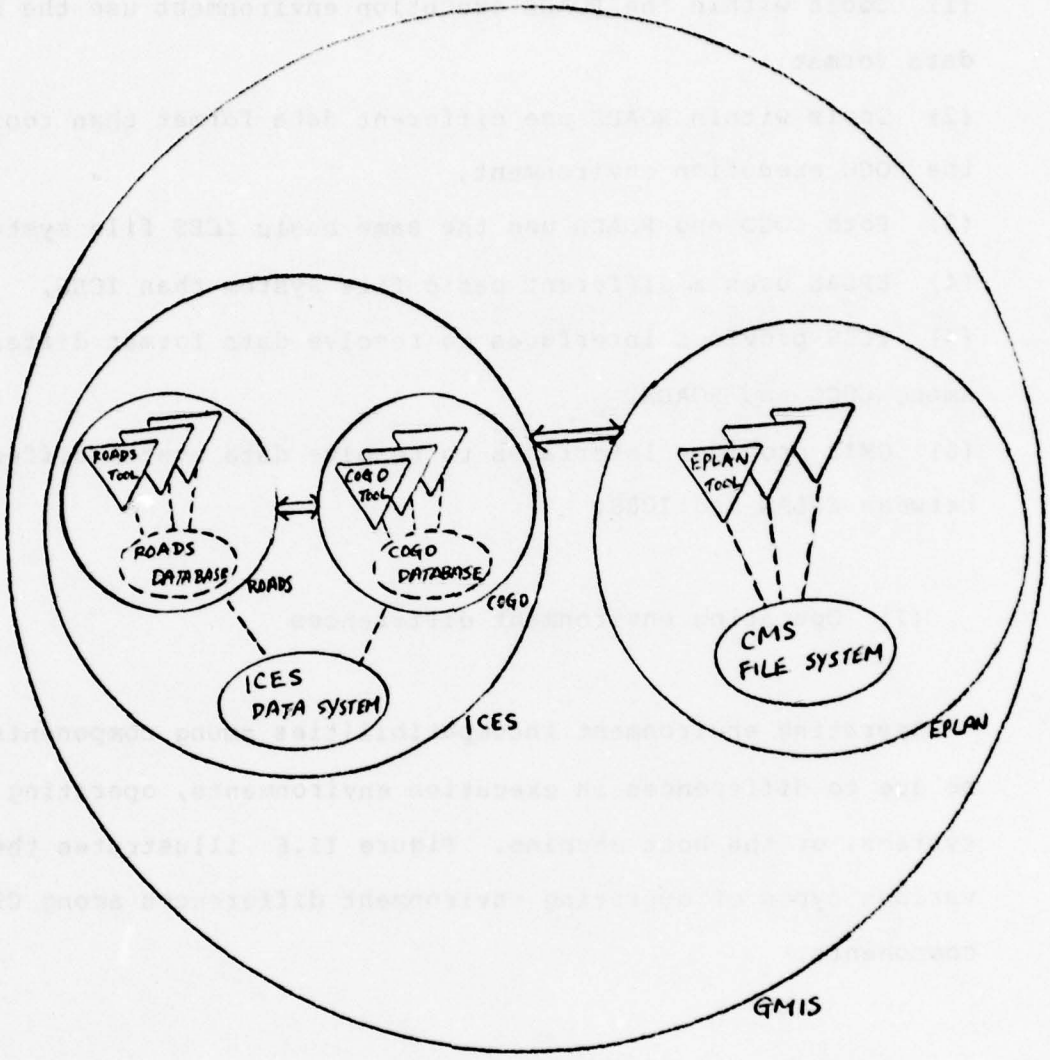


Figure II. 5

ILLUSTRATES DATA INCOMPATIBILITIES AMONG COMPONENTS

OPERATION ENVIRONMENT DIFFERENCES NOT SHOWN

and dotted ovals represent local data with the same format. Figure II.5 illustrates the following (The ICES, EPLAN, and GMIS systems are explained in later sections):

(1) Tools within the ROADS execution environment use the same data format,

(2) Tools within ROADS use different data format than tools in the COGO execution environment,

(3) Both COGO and ROADS use the same basic ICES file system,

(4) EPLAN uses a different basic file system than ICES,

(5) ICES provides interfaces to resolve data format differences among COGO and ROADS,

(6) GMIS provides interfaces to resolve data system differences between EPLAN and ICES.

(2) Operating environment differences

Operating environment incompatibilities among components may be due to differences in execution environments, operating systems, or the host machine. Figure II.6 illustrates the various types of operating environment differences among CIS components.

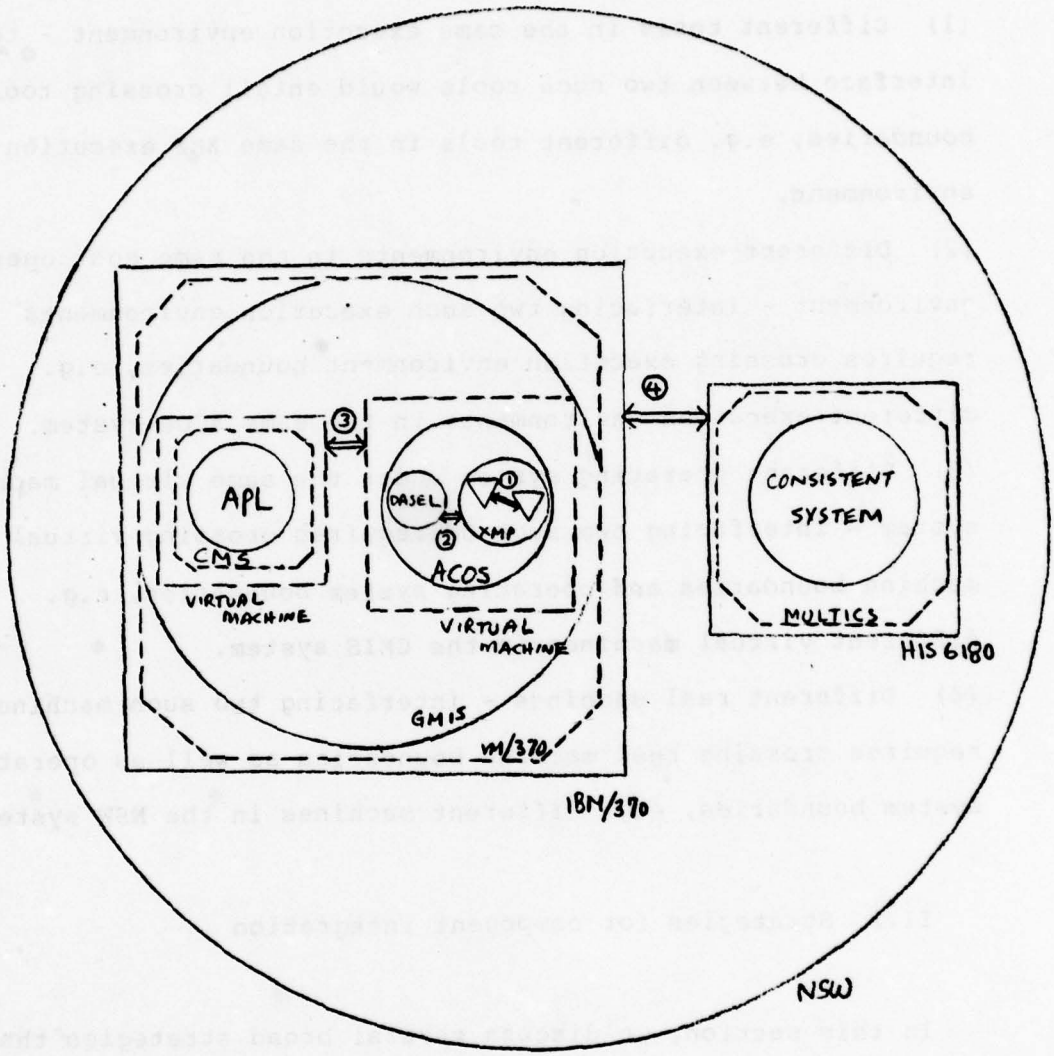


Figure II.6

ILLUSTRATES OPERATING ENVIRONMENT INCOMPATIBILITES AMONG COMPONENTS

(DATA INCOMPATIBILITES NOT SHOWN)

In Figure II.6, solid squares represent real machines, dotted squares represent virtual machines, and dotted octagons represent operating systems. Figure II.6 illustrates the following types of system boundaries:

(1) Different tools in the same execution environment - to interface between two such tools would entail crossing tool boundaries, e.g. different tools in the same XMP execution environment.

(2) Different execution environments in the same host operating environment - interfacing two such execution environments requires crossing execution environment boundaries, e.g. different execution environments in the same ACOS system.

(3) Different operating system under the same virtual machine system - interfacing two such OS requires crossing virtual machine boundaries and operating system boundaries, e.g. different virtual machines in the GMIS system.

(4) Different real machines - interfacing two such machines requires crossing real machine boundaries as well as operating system boundaries, e.g. different machines in the NSW system.

II.2 Strategies for component integration

In this section, we discuss several broad strategies that may be used for integrating CIS components, then we examine types of databases that may be used to support these strategies. There are two basic types of strategies for component integration, those used for resolving data differences and those used for

resolving system differences.

(1) Strategies used for resolving data differences

A frequently used strategy for resolving data differences among CIS components is to make use of a common data system. Data translation routines are provided by the CIS to move data from a component to the common data system, and from the common data system to a component. Thus the common data system, in effect, acts as an intermediary among components that use different data formats or data systems.

An alternative strategy for resolving data differences among CIS components is to make use of an interface routine for a pair of components. The number of interface routines required grows as the squared of the number of components in the system. For systems with small numbers of components, this strategy may be very effective since each interface routine is tailored for a particular component pair. These two strategies are not mutually exclusive, and a CIS may make use of both strategies.

(2) Strategies for resolving system differences

The basic problems here are communications and data transfer across system boundaries. To resolve these problems, three complementary strategies may be used. First, make use of existing, or develop new inter-process or inter-machine

communications protocol. Second, make use of a overseer process for coordinating these activities. Third, make use of special routines to encapsulate components, so as to trap the I/O issued by the components. A study of these strategies in a virtual machine environment is reported in (Lam and Madnick 78a).

(3) Databases for supporting strategies for component integration

We found three types of databases that may be used to support component integration in a CIS. First, static information about components in the CIS, e.g. catalog of the common file system, and catalog of the command syntax for certain components. Second, operating status of components in the CIS, e.g. current status of components being used. This type of information is used by the CIS overseer process in coordinating activities in the CIS. Third, detail information about components, e.g. their I/O characteristics and data requirements. These information can be used for constructing automatic interfaces among components.

III Study of specific systems that exhibit the CIS approach

A CIS provides a facility for joint usage of existing, often incompatible components by using various strategies and mechanisms to resolve the possible data/system incompatibilities among components. A CIS approach system development effort is one in which some characteristics of a CIS are present.

A literature study has been conducted to identify information systems that exhibit the CIS approach. Eight generic types of CIS were identified using a taxonomy of CIS according to their structural characteristics. The nine example CIS-type systems that we identified in the literature fall into the various generic types, as shown in Figure III.1.

In each of the subsections below, we shall briefly discuss each generic type of CIS and the sample systems within it.

III.1 Type I - Cross-tool-different-data-format

The problem of different tools using different data formats even when these tools operate under the same operating environment arise very often. This is one reason for the development of special execution environments whereby tools that are often used for an application area are brought under a consistent operating environment using compatible data formats. We shall briefly describe one such system, called Data Analysis

	DIFFERENT DATA FORMAT	DIFFERENT DATA SYSTEM AND DIFFERENT DATA FORMAT
CROSS TOOL BOUNDARY	I DAS	II VSAM/ISAM ARTHUR ANDERSON
CROSS EXECUTION ENVIRONMENT BOUNDARY	III ICES ACOS	IV CONSISTENT SYSTEM DAISY
CROSS OS BOUNDARY	V	VI GMIS
CROSS MACHINE BOUNDARY	VII	VIII NSW

Figure III.1

Categorization of sample CIS-type systems

System (DAS) (Levitt et. al. 74).

The Data Analysis System (DAS) was developed at the RAND Corporation to provide a flexible data analysis system with advanced graphic displays. This effort was initiated to meet the need for flexible statistical tools required in the process of solving complex statistical problems. The system was seen as a solution to the difficulties of using current statistical packages, since most current 'canned' statistical analysis programs are very rigid in their data requirements, thus place a burden on the user to repeatedly restructuring data when there is a need to use several of these programs in succession.

Figure III.1.1 illustrates the structure of DAS. Data analysis programs in DAS include FORTRAN programs developed in-house, as well as subroutines acquired from the BIOMED library. These routines all operate under the same OS/MVT operating system. A common DAS database, called Data Analysis Data Base (DADB) is used by all the programs in DAS. Data in the DADB is organized into sets. A set is a table of values. Operations on sets include editing, subsetting, intersection, union, etc. DAS was implemented on an OS/MVT system on an IBM/360 machine. A graphic subsystem was used to make on-line access to DAS possible.

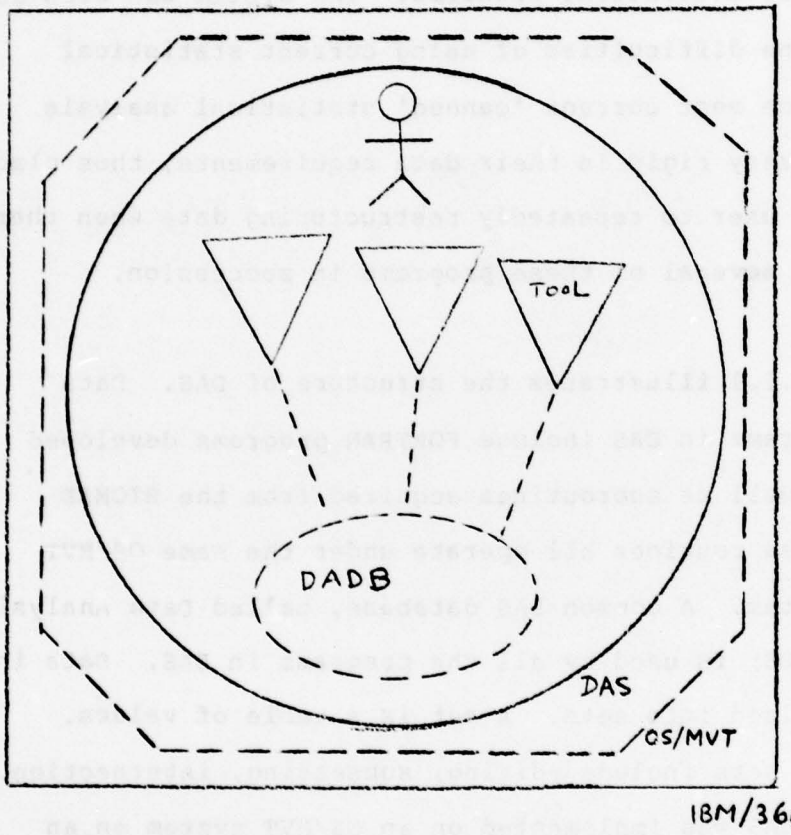


Figure III.1.1 DAS

III.2 Type II - Cross-tool-different-data-system

Under the same operating environment, tools may use different data systems. For example, it is quite common that two different database systems be operating under the same operating system. Interfacing tools that use different data systems involves crossing these data system boundaries. Let us examine two such cases.

(1) ISAM/VSAM interface

The IBM Index Sequential Access Method (ISAM) has been widely used. ISAM organizes files in physical cylinders. One track on the cylinder is reserved for storing the indices, one for each data track on the cylinder. There is a separate area to store higher levels of indices for these track indices. Data on a track is organized in sequence with respect to a key. The index entry correponding to a data track indicates the highest key on that track. This arrangement of data in the file allows an user random access to a record based on a key and access the records in key sequence efficiently. Two major problems with ISAM are: (1) when no more records can fit on a track the overflow records are stored in an overflow area and chained in key sequence, thus deteriorates access time, and (2) there is no dynamic space reclaimation for deleted records.

The Virtual Storage Access Method (VSAM) was developed by IBM

as an improvement over ISAM. VSAM has similar overall logical file organization as ISAM but organizes the file in logical cylinders called CONTROL-AREAs and logical tracks called CONTROL-INTERVALs. By allowing CONTROL-AREA and CONTROL-INTERVAL splitting when they become full, VSAM avoids using any overflow area. The space within a CONTROL-INTERVAL is managed dynamically so that the space occupied by deleted record can be reclaimed.

An interface program was developed by IBM so that organizations with existing programs that use ISAM can continue to operate without modification, when the ISAM dataset is converted into a VSAM dataset. Figure III.2.1 illustrates this situation before the interface is used and Figure III.2.2 illustrates the situation after the interface is used. Presumably new programs would be written directly for VSAM bypassing the ISAM/VSAM interface.

This is one of the examples of a more general problem. A major investment is made in the application programs that are based on a particular database system or file system. If a major improvement over these file system and database system becomes available, to ease the transition to the new technology, an interface is developed so that existing programs can still operate on the old data system while actually the data is managed by the new data system.

(2) Arthur Anderson's Data Dictionary

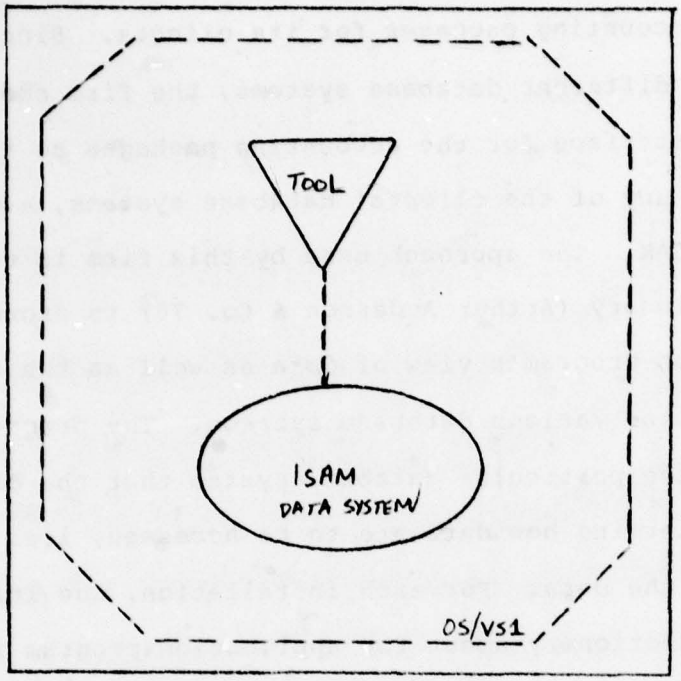


Figure III.2.1 BEFORE USING INTERFACE

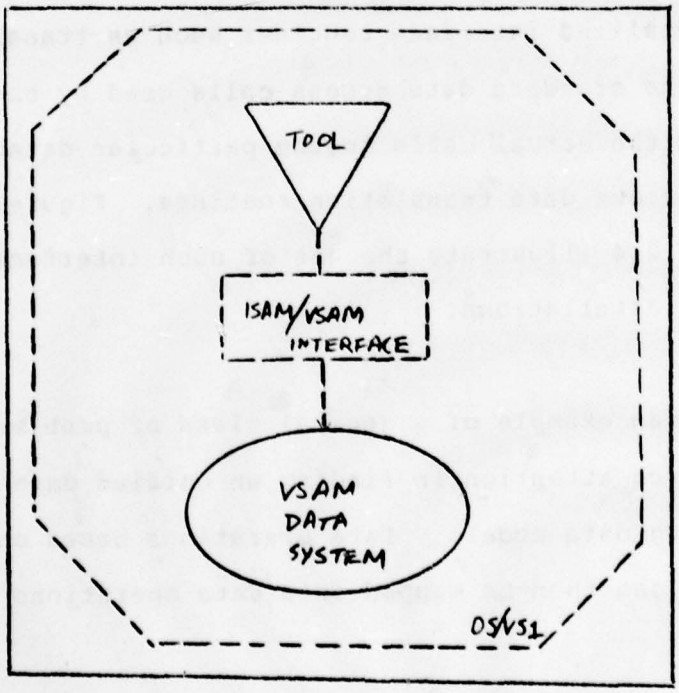


Figure III.2.2 AFTER USING INTERFACE

Arthur Anderson & Co., a major accounting firm, developed several accounting packages for its clients. Since these clients often use different database systems, the firm chose to develop a general interface for the accounting packages so that they can be used with any of the clients' database systems, e.g. IMS, IDMS, ISAM or VSAM. The approach used by this firm is to make use of a data dictionary (Arthur Anderson & Co. 76) to store the application program's view of data as well as the definition of data on these various database systems. The program's view of data and the particular database system that the data is actually stored determine how data are to be accessed, i.e. the procedure to access the data. For each installation, the information in the data dictionary about the application program is used to create a database I/O interface program for that application program for the particular database system used by that installation. This interface then acts as a coordinator to call other generalized interface routines such as translation routines that map the standard data access calls used by the application program to the actual calls to the particular database system, and the various data translation routines. Figure III.2.3 and Figure III.2.4 illustrate the use of such interfaces in two different installations.

This is an example of a general class of problems. There is much research attention in finding an unified data model for all the existing data models. Data operations based on this unified data model can then be mapped into data operations based on any

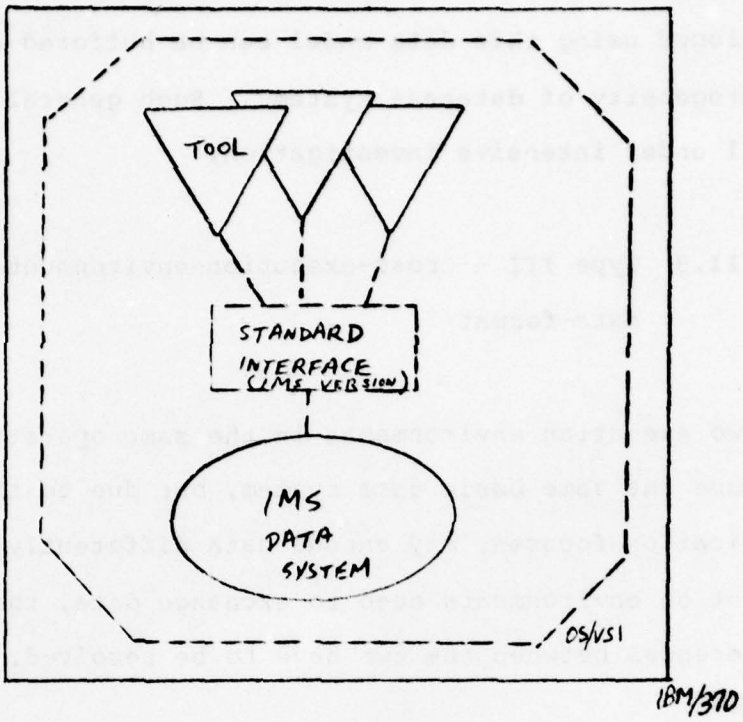


Figure III.2.3 INSTALLATION i

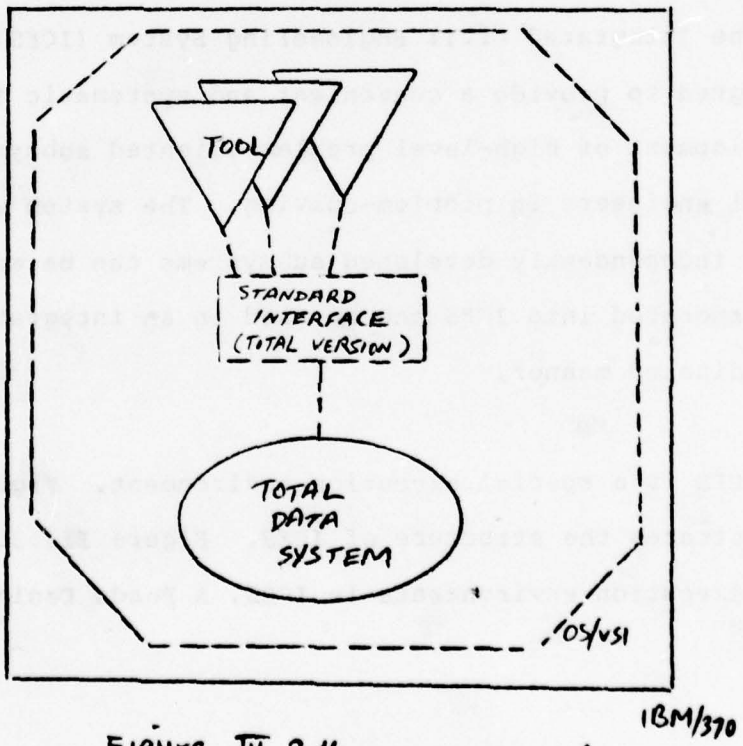


Figure III.2.4 INSTALLATION j

existing or future data models. Thus application programs developed using this data model can be buffered against the heterogeneity of database systems. Such general solutions are still under intensive investigation.

III.3 Type III - Cross-execution-environment-different-data-format

Two execution environments in the same operating environment may use the same basic data system, but due to their different application focuses, may encode data differently. When these two execution environments need to exchange data, these data format differences between the two have to be resolved.

(1) ICES

The Integrated Civil Engineering System (ICES) (Roos 67) was designed to provide a convenient and systematic facility for development of high-level problem-oriented subsystems to aid civil engineers in problem-solving. The system was designed so that independently developed subsystems can be easily incorporated into ICES and be used in an integrated and coordinated manner.

ICES is a special execution environment. Figure III.3.1 illustrates the structure of ICES. Figure III.3.1 illustrates two execution environments in ICES, A Roads Design System

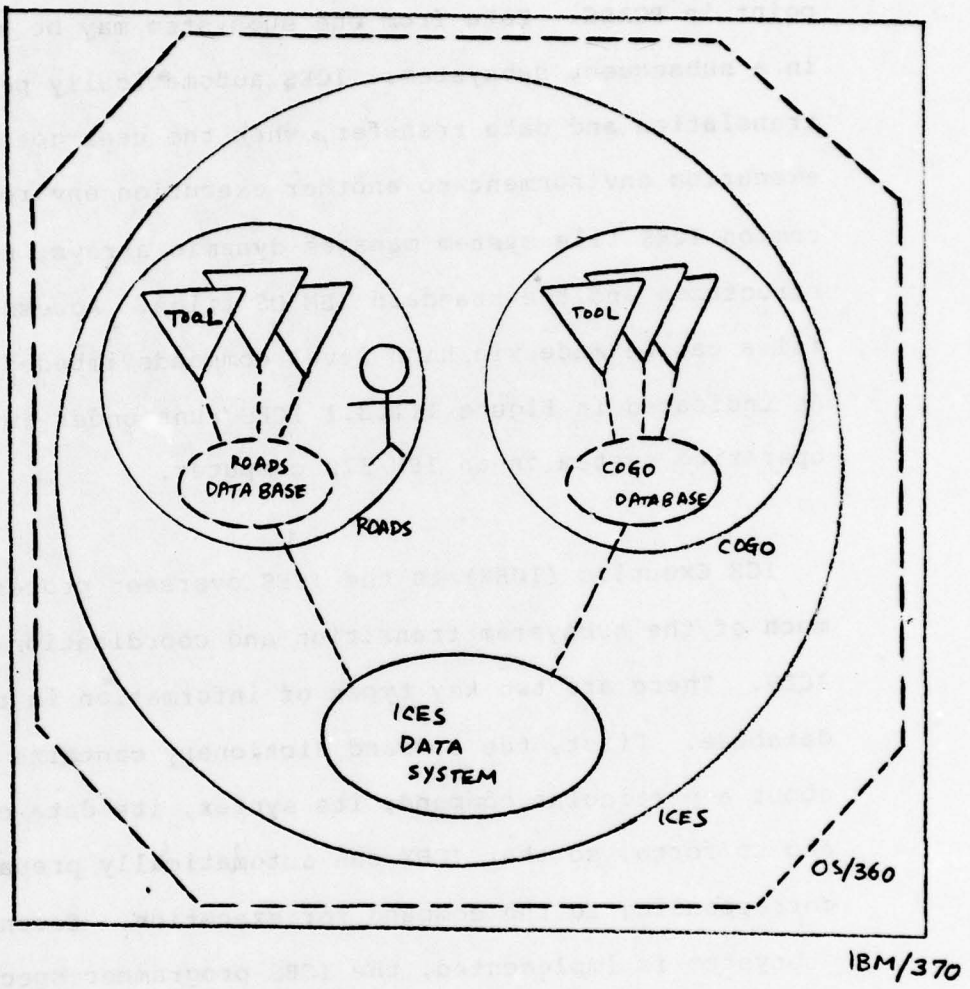


Figure III.3.1 ICES

(ROADS), and a Coordinate Geometry System (COGO). Both execution environments use the same basic ICES file system but each uses different data formats. For example, a point in COGO may be represented quite differently from the representation of the same point in ROADS. Data from one subsystem may be saved to be used in a subsequent subsystem. ICES automatically performs the data translation and data transfer, when the user goes from one execution environment to another execution environment. The common ICES file system manages dynamic arrays, relational data structures and the standard IBM OS files. Access to any of these files can be made via high level commands imbeded in programs. As indicated in Figure III.3.1 ICES runs under an OS/360 operating system on an IBM/370 computer.

ICE Excutive (ICEX) is the ICES overseer process that perform much of the subsystem transition and coordination activities in ICES. There are two key types of information in the ICES system database. First, the command dictionary contains information about a particular command, its syntax, its data characteristics, and so forth, so that ICEX can automatically prepare the module corresponding to the command for execution. Second, When a subsystem is implemented, the ICES programmer specifies the characteristics of the data structures required in the subsystem. This information is stored in the system dictionary. The data translator uses this information to create the corresponding new data structures in a new subsystem when a user request data transfer to that subsystem.

(2) ACOS/ACOL

The Application Control Operating System/Application Control Language (ACOS/ACOL) (NBER 74) (NBER 75a) was developed by the National Bureau of Economic Research (NBER) Inc.. The system was designed to host existing and future NBER developed tools. These tools include the Time-Shared Reactive On-line Laboratory (TROLL), which has been widely used for econometric and data analysis.

Figure III.3.2 illustrates two execution environments in ACOS/ACOL. These are the Experimental Mathematical Programming System (XMP), and the Data Analysis and Statistical Experimental Language (DASEL). XMP is an advanced mathematical programming system. DASEL is a powerful high-level data analysis language that operates on generalized arrays. Both execution environments, as well as other execution environments in ACOS/ACOL, use the same ACOS/ACOL file system. This common file system is structured much like the MULTICS hierarchical file system. The execution environments may encode data very different from one another. For example, DASEL deals with generalized arrays while XMP deals with special data files convenient for mathematical programming.

As illustrated in Figure III.3.2, ACOS operates as an operating system for a virtual machine under the VM/370 operating system. The VM/370 operating system runs on an IBM/370 computer.

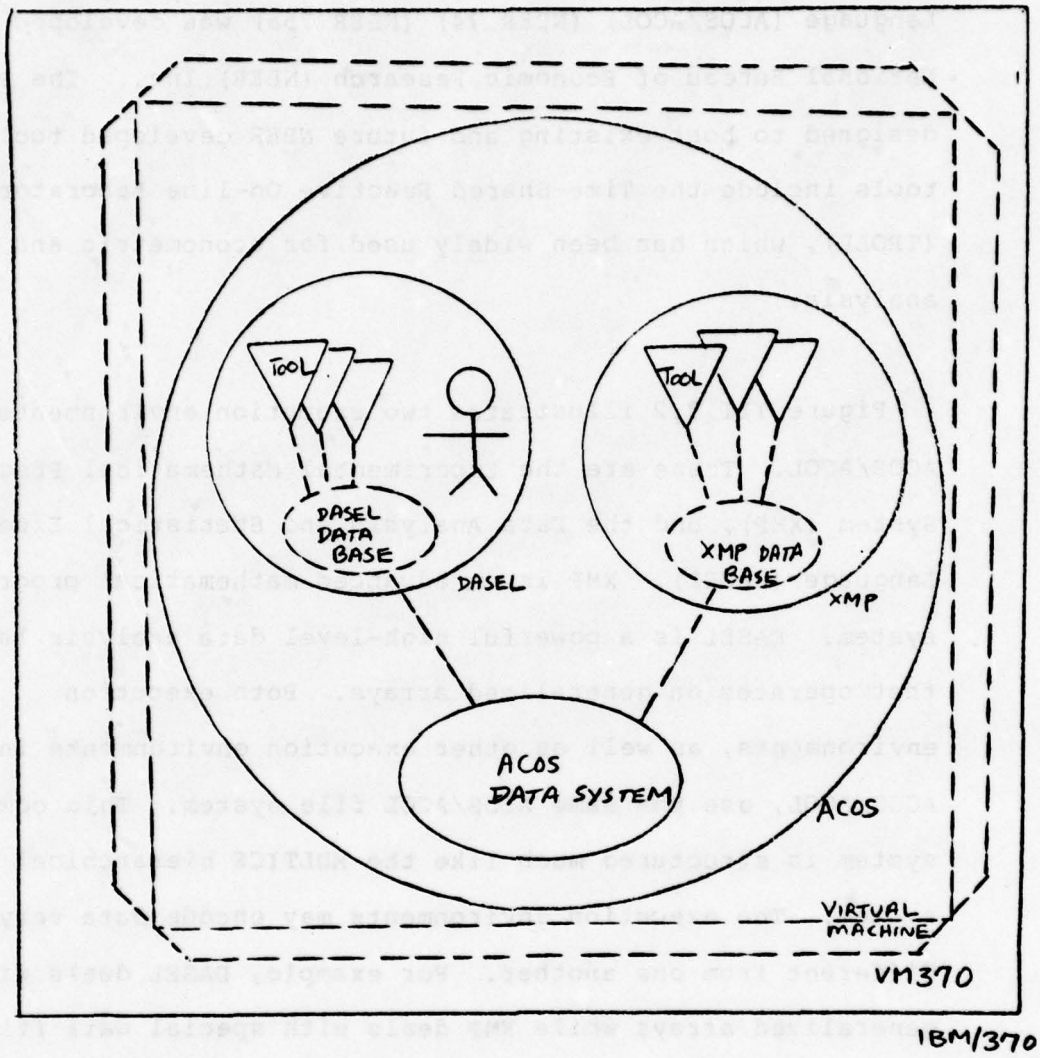


Figure III. 3.2 ACOS/ACOL

Thus ACOS performs all the functions that an operating system usually performs, such as handling I/O, storage management, program initation and termination, and managing the associated databases to support these functions. ACOL is the control language for ACOS. It is a very high-level macro language for specifying the sequences of operations to be performed by ACOS. For example, when a tool is created, a programmer can use the ACOL facility to specify the syntax of the command that can be used to invoke the tool, the various I/O parameters associated with running the tool, and the sequence of operations that ACOS performs to initiate the tool. This information is processed and stored in a database to be used by ACOS in responding to users' requests.

III.4 Type IV - Cross-execution-environment-different-data-system

Two execution environments in the same operating environment may use different data systems. Interfacing between these execution environments involves crossing data system boundaries.

(1) The Consistent System

The Cambridge Project was a cooperative effort organized by scientists at M.I.T. and Harvard University with the goal of making the computer more useful and usable by researchers in the basic and applied behavioral sciences, and in other sciences that

have similar computing problems. The project had two main goals: first, to develop the necessary programs and computing tools; and second, to combine these tools and others that may be developed elsewhere into a consistent set of programs, models, and data. The Consistent System (CS) is part of the efforts of the Cambridge Project.

The Consistent System (CS) is a special execution environment. Figure III.4.1 illustrates three execution enviornments within CS. These are the JANUS database system (Cambridge Project 74b), the Time Series Processor (TSP), and the Mathematical Programming system (MP) (Cambridge Project 74a). All three execution environments use different data systems. JANUS is an user-oriented database system that enables the user to manipulate data as logical entities and relationships among the entities using a high-level query language. TSP is a modified version of the popular batch TSP programs that use their own data system. MP is developed for the CS and uses the CS file system. The CS file system is used as an intermediary among these different data systems. For example, a user may enter JANUS, perform some data management functions on JANUS files, select specific results of interest and put them in a CS file, leave JANUS, go back to CS listener level, enter TSP and issue a TSP command to get the CS file created in JANUS for further analysis in TSP. There are several major types of CS files. These are the CHARACTER files, the MN-arrays, and the GENERAL-arrays. Each file consists of a file description and a data portion. All programs in CS can

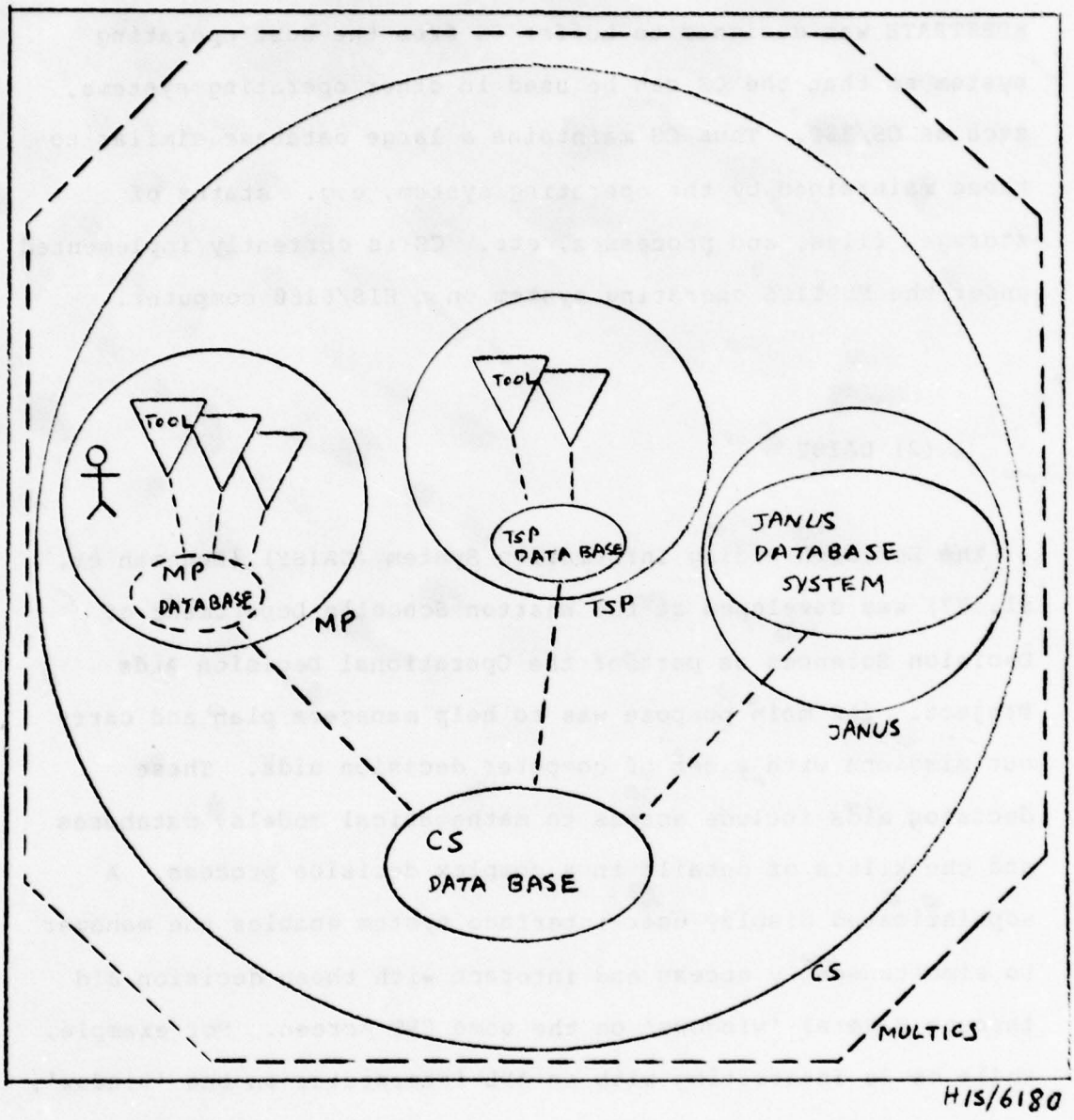


Figure III.4.1 THE CONSISTENT SYSTEM

accept CS files as input and can output CS files.

The control of CS is mainly the task of the CS SUBSTRATE. The SUBSTRATE was designed to buffer CS from the host operating system so that the CS can be used in other operating systems, such as OS/360. Thus CS maintains a large database similar to those maintained by the operating system, e.g. status of storage, files, and processes, etc. CS is currently implemented under the MULTICS operating system on a HIS/6180 computer.

(2) DAISY

The Decision Aiding Information System (DAISY) (Buneman et. al. 77) was developed at the Wharton School's Department of Decision Sciences as part of the Operational Decision Aids Project. Its main purpose was to help managers plan and carry out missions with a set of computer decision aids. These decision aids include access to mathematical models, databases and checklists of details in a complex decision process. A sophisticated display user-interface system enables the manager to simultaneously access and interact with these decision aid through several 'windows' on the same CRT screen. For example, while he is interacting with an APL interpreter on one 'window', another window may be displaying data retrieved from a WAND database system. This 'window' mechanism can be an effective integration tool to bring together multiple subsystems

simultaneously to aid an decision.

The primary users of the DAISY system are the tactical managers who interact with the decision aids, such as models, and database systems. Figure III.4.2 illustrates an user simultaneously interacting with an APL execution environment and the WAND database system. The WAND database system structures data as logical networks (CODASYL 71). APL uses data in the form of vectors and matrices. There is an automatic model interface mechanism in DAISY. When a model requires some data, DAISY automatically formulates the necessary data access queries to the appropriate database systems to obtain the data.

To accomplish the above automatic model interface, DAISY requires that new models to be incorporated into DAISY following certain standard rules about I/O. To each model is added a 'front-end' and a 'back-end' interface to handle the data needs of the model. A model interface program is used to obtain data from the database systems before the model is initiated. When the model is initiated, the 'front-end' calls a routine to translate the data from the model interface into a form usable by the model. The function of the 'back-end' is exactly the reverse, it calls a data translation routine to put the data in a file usable by the model interface, which then passes the data to other parts of the DAISY system. Data transfer and communication among the components in the DAISY system is mainly using a inter-process logical communications mechansim.

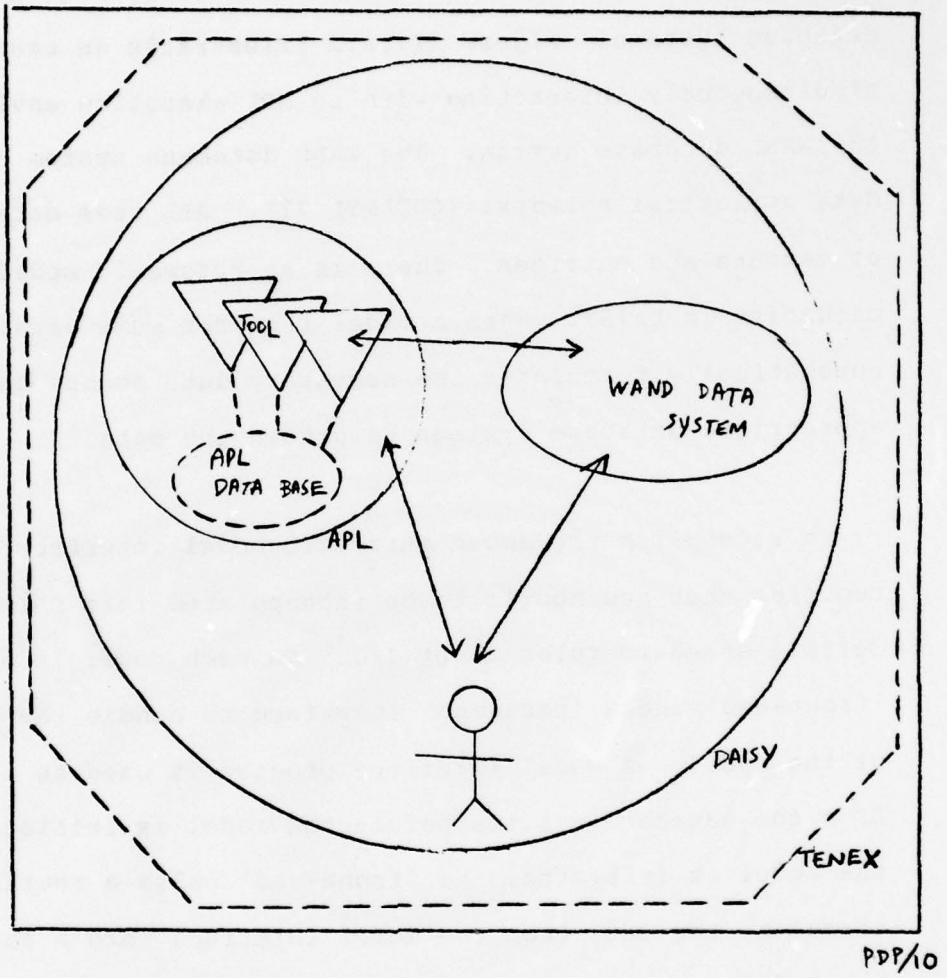


Figure III.4.2 DAISY

DAISY is implemented under a TENEX operating system on a PDP-10 computer. The coordination of the various DAISY processes is performed by the DAISY executive, which includes the window manager, the DAISY control program and the model interface program. The system database used by DAISY, includes among other information, a dictionary of models in the DAISY environment. Each model description in the dictionary contains information about the type of data the model requires, the sources of the data, and the data format that the model will accept. Using this database, the model interface can automatically generate data access requests to the appropriate database systems.

III.5 Type V - Cross-OS-different-data-format

Several different operating systems may coexist in a virtual machine operating environment, each residing on a virtual machine. Two components on these different virtual machines may use the same type of file system but encode the data differently. For example, a component in an OS/VS1 operating system on one virtual machine and another component in an OS/360 operating system on another virtual machine may use the same type of OS/360 file system, but may encode data differently to suit each type of application that the component serves.

III.6 Type VI - Cross-OS-different-data-system

The Generalized Management Information System (GMIS) was

designed by the joint efforts of the MIT Sloan School of Management and the IBM Cambridge Scientific Center. It was initially designed to serve as a base for decision support systems to aid energy impact analysis and policy making, as part of the New England Energy Management Information System (NEEMIS) Project. The GMIS architecture exploited virtual machine technology (Donovan and Madnick 76a) to provide a facility that allows rapid integration of computer programs and databases.

Figure III.6.1 illustrates two execution environments in GMIS, APL and SEQUEL. APL is a generalized language environment operating under the CMS operating system on a virtual machine. SEQUEL (Chamberlin and Boyce 74) is a relational database system operating on another virtual machine. SEQUEL manages its own file system and its own I/O operations. An user performing analysis in an APL execution environment may request data from the SEQUEL database system. The data is sent from the SEQUEL database system to the user's APL work space, available for further analysis.

GMIS makes use of several strategies for communications among virtual machines and for resolving data format differences among the user's execution environments and the various database systems. These strategies include: use of shared virtual disk for data transfer, use of direct mapping between virtual card reader and virtual card punch for synchronization, and use of inter-virtual machine communications via virtual processor

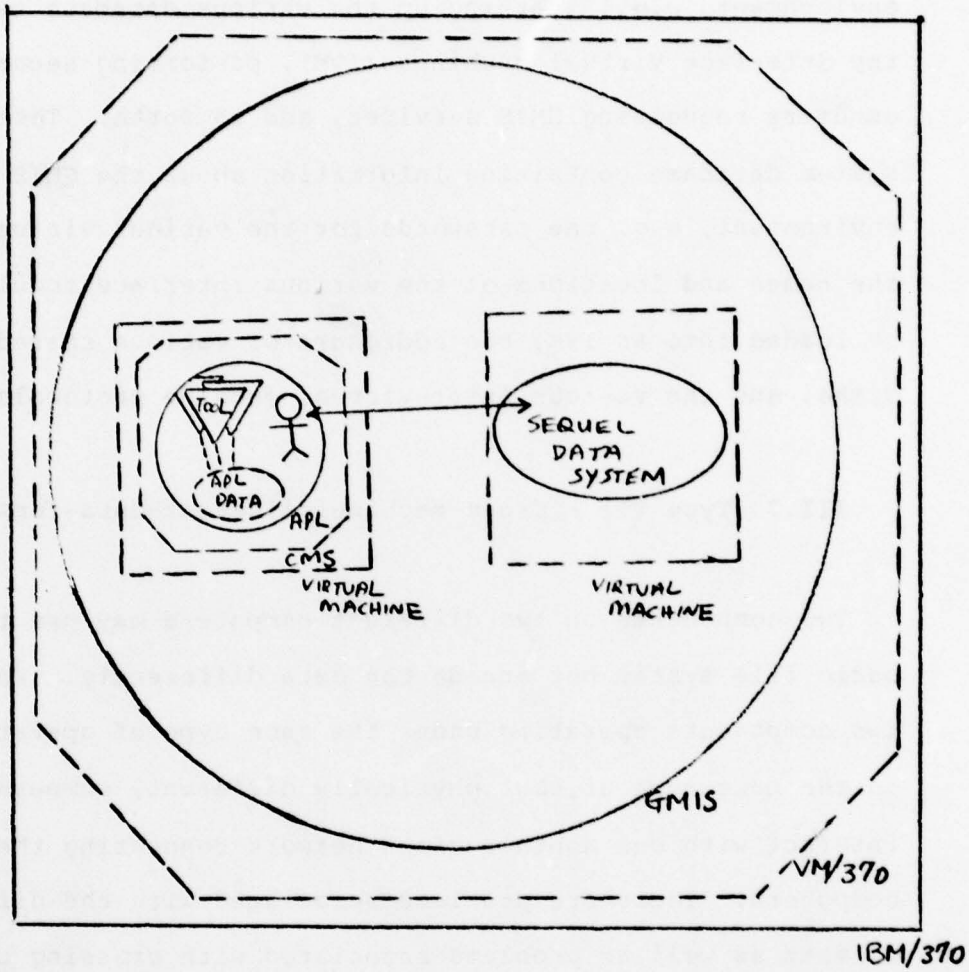


figure III.6.1

interrupt. See (Lam and Madnick 78b) for a detail discussion of these mechanisms.

There is a special virtual machine in GMIS, called the Manager Virtual Machine (MVM), that handles the initiation of the GMIS environment, e.g., starting up the various database systems and the Interface Virtual Machines (IVM), performing security checks on users requesting GMIS services, and so forth. There is a system database containing information about the GMIS environment, e.g. the passwords for the various virtual machines, the names and locations of the various interface modules that may be loaded into an IVM, the addresses of various shared virtual disks, and the various inter-virtual machine protocols.

III.7 Type VII - Cross-machine-different-data-format

Two components on two different computers may use the same basic file system but encode the data differently. For example, two components operating under the same type of operating system, on the same type of, but physically different, computers may interact with one another via a network connecting the two computers. There are problems associated with the different data formats as well as problems associated with crossing computer boundaries.

III.8 Type VIII - Cross-machine-different-data-system

Components may operate under various operating systems on different computers using different file systems. Problems associated with crossing data system boundaries as well as computer boundaries have to be solved when these components interact with one another.

(1) NSW

The National Software Works (NSW) (MCA 76) is an effort to provide for the convenient coordination and execution of programs, operating on physically incompatible and geographically distributed computers. NSW makes use of the ARPANET computer network for communication among the various host computers that participate in the project. NSW minimizes the expensive software replication efforts by allowing programs on different computer systems to be used by an user at any particular computer system. Figure III.8.1 illustrates an user in the APL execution environment interacting with the Consistent System. The APL execution environment operates under an OS/VS1 operating system on an IBM/370 computer at UCLA. The Consistent System is under MULTICS on a HIS/6180 computer at MIT. Since these two execution environments use entirely different data systems on different computer systems, NSW has to resolve these incompatibilities. NSW makes use of translation routines called File-Packages for these purposes. For example, to send a MULTICS file to an

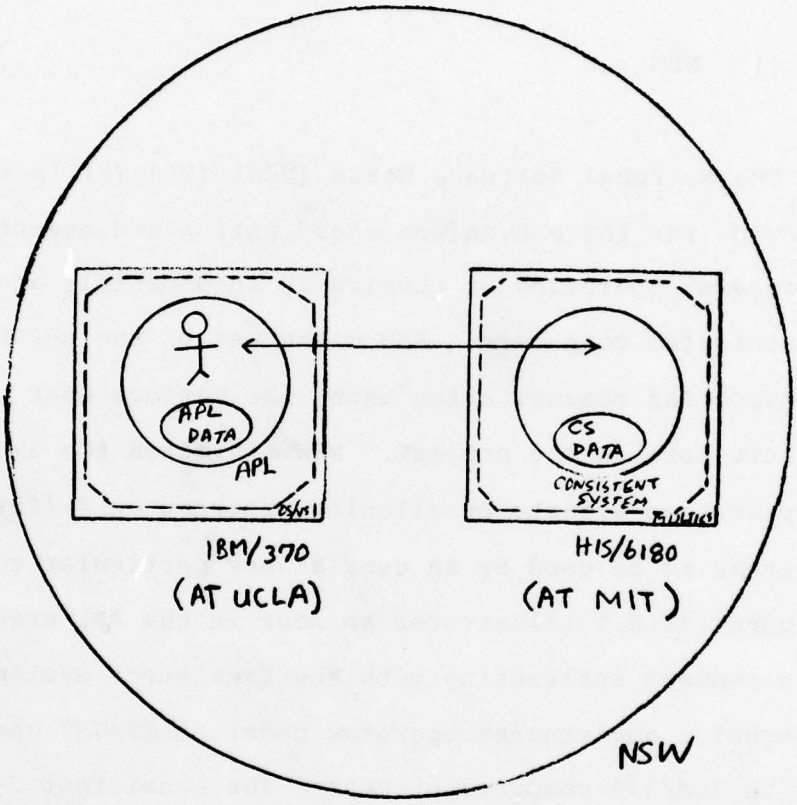


Figure III.8.1 NSW

IBM/370, the File-Package on the MULTICS translates the file into a standard form, and sends it to the IBM/370. The File-Package on the IBM/370 then translates the file in standard form into a format usable by the IBM/370.

NSW maintains a common file system. The files in the NSW file system are composed of files from various host operating systems, for example, part of the file system may be maintained on the MULTICS system while part of it may be on the OS/VS1. The user is not aware of the physical locations of the files. There are utilities for importing a file on any host system into the NSW file system and for exporting a NSW file to any host system.

The key components of the NSW control mechanism include the MSG communication protocol, the Works Manager (master coordinator of the NSW), the Front-End (one on each site to communicate with the terminal user), and the Foreman (one on each site that cooperates with the host operating system to run the program requested by a remote user). Typically, an user requests NSW services via the Works Manager. The Works Manager passes the request to the appropriate Foreman which starts the program for the request. The Foreman then communicates with the Front-End to obtain input from the user and send output to the user. The NSW Works Manager maintains a directory of information about each program in the NSW environment, e.g. location of the program. The Works Manager is also responsible for maintaining a file catalog for the NSW file system, and an user catalog describing

the access rights and user accounting information.

IV Conclusions

We have attempted to clarify and define a relatively new concept in Information Systems architecture, that of Composite Information Systems. A structural model of CIS has been developed and used to study several existing information system that exhibit the CIS approach. The study shows that architecturally, CIS can be categorized into 8 generic types according to the types of data and system incompatibilities among the CIS components.

Several areas of CIS require further investigation. First, study of machanisms used for resolving component incompatibilities. Second, study performance implications of these mechanisms. Third, study mechanisms for facilitating automatic component interfaces. Fourth, development of a functional model of CIS, i.e. study of CIS from the point of view of the functional types of interfaces provided by the CIS.

PRECEDING PAGE BLANK

Addendum Technical Report 4

Preliminary Investigations of a SPO Decision Support System

Chat-Yu Lam

Stuart E. Madnick

Preliminary investigations of a SPO DSS

I Introduction

The contents presented in this report are based on preliminary examinations of the available literature and informal discussions regarding the organization and function of Air Force System Program Offices (SPOs). Major functions in an Air Force SPO are discussed in section II. Section III identifies characteristics and examples of SPO decision problems for which a Decision Support System (DSS) is likely to be of great value. The existing resources within SPO as well as potential resources that may be available for a SPO Decision Support System are identified in section IV. Section V introduces the concept of a Composite Information System as an approach to realize a SPO Decision Support System and to aid ad-hoc SPO decision-making.

II. SPO organization

A SPO is organized with the objective of technically and financially managing a system development effort. A SPO exists for the duration of a system development effort within an Air Force organization. It is the director and coordinator of the various development efforts by the contractors and those of the SPO personnel.

A SPO usually consists of 50 to 200 persons. It consists of the SPO top management and the various functional divisions. The major functional divisions are : (1) data control, (2) quality control, and (3) engineering. Each functional division may have personnel from other parts of the Air Force organization. Figure II.1 illustrates a simplified matrix organization of a SPO. The SPO management tasks and tasks performed within each of the major functional divisions are briefly discussed below.

II.1 SPO top management tasks

(1) Develop project plan - A SPO has to develop a plan for the development project. This plan is for the entire development period and may range from 3 to 12 years. To provide cost estimates for the plan, a SPO uses the cost estimates for each subtask of the development project provided by the contractors and its own financial support organization. SPO personnel also supports an independent cost estimate of the project performed by

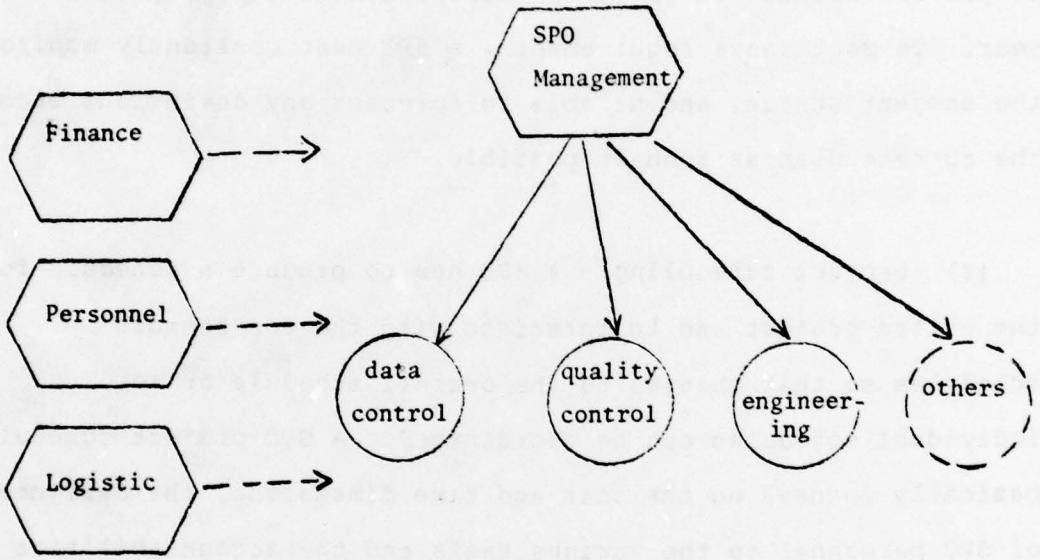


Figure II.1 Illustration of matrix organization of a SPO

the funding agency.

(2) Yearly budgeting - Each year, funds for the project for that year are acquired from the funding agency. A SPO is required to supply details of current project status, such as costs, tasks performed, and project schedule with respect to the planned schedule (Zero Base Budgeting). A SPO is also required to provide details on projected milestones of the project for the year. To meet these requirements, a SPO must constantly monitor the project status, and be able to forecast any deviations from the current plan as soon as possible.

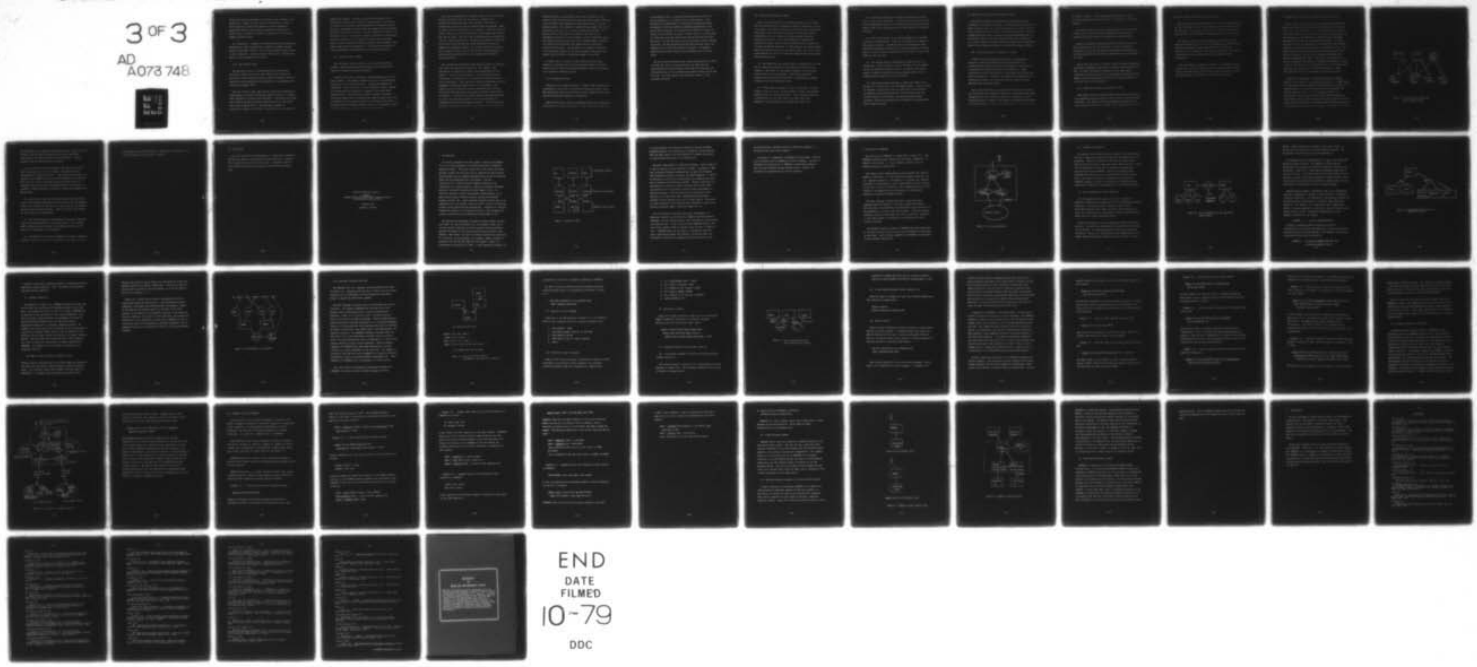
(3) Project scheduling - A SPO has to produce a schedule for the entire project and to interface with the contractors' schedules so that changes to the overall schedule or to individual schedules can be coordinated. A SPO project schedule basically focuses on the cost and time dimensions, the assignment of SPO personnel to the various tasks and the accountabilities of these personnel typically has not been integrated into the project schedule or budget in a systematic fashion.

(4) Project control - To evaluate actual project performance against planned project performance, a SPO interfaces with the contractors' reporting systems to obtain an overall project report. This interface is usually manual.

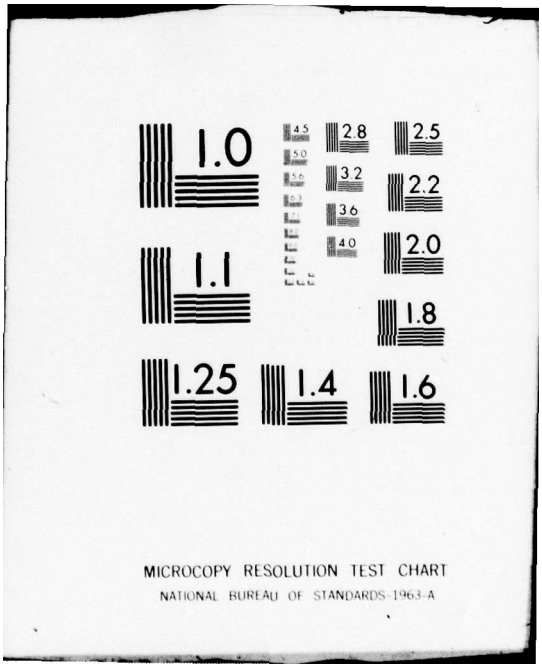
(5) Project forecasting - A SPO has access to substantial

AD-A073 748 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
USE OF VIRTUAL MACHINES IN THE DEVELOPMENT OF DECISION SUPPORT --ETC(U)
JUL 79 J J DONOVAN, S E MADNICK, C LAM F30602-77-C-0205
UNCLASSIFIED RADC-TR-79-187 NL

3 OF 3
AD A073 748



END
DATE FILMED
10-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

amounts of historic information on project costs, schedule, and performance. Based on these and other information, a SPO is often required to assess the impact of certain changes to engineering design, and/or project schedule on the current project plan. Project forecasting is also part of the yearly budget planning process.

(6) Replanning - Changes are standard rather than exceptional phenomena in a SPO. Changes may be induced by budget changes, project requirement changes, cost overrun, problem areas, as well as new innovations. Changes include redirection in engineering design, project schedule, and costs.

II.2 Data control tasks

The SPO Data Control is an administrative function that maintains a master file of all data coming from contractors. This data includes monthly financial reports, periodic technical reports on work accomplished and planned for, and proposals for engineering changes (ECP).

Over the life of a SPO, Data Control would have accumulated a large amount of valuable information about the project. However, this information is usually kept manually. Contractor data comes into a SPO in the form of manual reports which are then filed into a set of master files. In many cases, reports or engineering changes are intended to supersede previous reports or

engineering changes. In most current manual systems, these materials are often kept strictly in chronological order making it difficult to determine which documents are actually relevant. Retrieval and analysis of these data for supporting unanticipated decisions can be more effective if this information is kept in an automatic system. An automatic system to manage this information will also facilitate the interface to various SPO management models, such as financial models and project process models, that may be used by SPO top management to aid critical decisions in various management tasks discussed in the previous section.

II.3 Quality control tasks

The SPO Quality Control maintains all current engineering designs, updates these engineering designs, and evaluates the impacts of engineering change proposals.

A SPO may have many contractors, each developing part of the final product. SPO keeps an updated version of all engineering designs of the various contractors. Engineering changes are inevitable. They arise due to cost or schedule changes, changes in the manufacturing process, or changes in the requirement specifications. An engineering change proceeds as follows: A contractor would submit an engineering change proposal (ECP) with a statement of the expected impact of this engineering change. A SPO would have to evaluate the ECP, and if it is favorable, issue an engineering change order (ECO) to the contractor.

The task of maintaining an updated version of all engineering designs consistent with all the ECPs is a complex one. Engineering designs and ECPs come from many contractors in unpredictable quantities and at unpredictable frequencies. Many of these ECPs are interrelated, for example, an ECP may supercede another ECP, or a group of ECPs may be inconsistent with another ECP, and so forth. Thus the task of maintaining the engineering file is a very difficult one. Furthermore, effects of an ECP on the performance parameters has to be evaluated by using the appropriate simulation models. The overall effect of an ECP as well as specific effects of an ECP on particular parameters of the engineering design have to be evaluated by SPO Quality Control personnel.

The information maintained by SPO Quality Control is required even after the project has completed. For example, this information is required to support the manufacturing of the product. At the time of production, the engineering design information is transferred to the logistic division of the Air Force organization. The logistic division is responsible for any subsequent engineering modifications to the product. The logistic division naturally is more concerned with the costs associated with maintaining the product, and the reliability of the product. Since there are often logistic personnel matrixed in a SPO, they represent another view of the engineering information kept by the SPO Quality Control. Since the logistic personnel is concerned with the implications of new engineering

desings and ECPs on maintenance costs and reliability, they evaluate ECPs from this point of view and often require different types of models than engineering simulation models or project process models. For example, a major ECP may suggest the use of alternative materials for a certain engineering design. SPO Quality Control may assess the impact of the ECP on the aerodynamic properties of the product, using a simulation model. The logistic personnel may use a cost tradeoff model to assess the production/maintenance cost implications of using the alternative materials. SPO management is primarily concerned with the overall impact on the entire project, and may use yet another type of model to assess the impact of this ECP.

It seems that an automatic system would be particularly effective in handling the information requirements of the SPO quality control tasks and provide different views of information to be used with different models.

II.4 Engineering tasks

Engineering is a major SPO task. A team of SPO engineers may be assigned to work with a contractor's engineering team. Such SPO engineering teams also provide a strong integrating force for engineering designs produced by contractors.

SPO often develop or acquire simulation models for controlling the quality of engineering designs. These simulation models may

be developed by SPO, or acquired from other parts of the Air Force organization, or acquired from a subcontractor. SPO engineers are responsible for running all engineering models. Any design or ECP that affects performance parameters of the system being seveloped is simulated using models to assess the impact of the engineering design. Hence, there is a strong need for effective interface between SPO engineering and other SPO functions. As most SPO data control and quality control systems are manual, the SPO engineering group often maintain their own separate file of ECPs and engineering design. An automatic system would eliminate delays and inconsistencies that often arise.

We have briefly discussed major tasks performed within a SPO. It seems that there is great potential for improving the effectiveness of decision-making within SPO by automating and integrating many of the information systems currently being used by a SPO. We shall explore this potential further in the following sections.

III SPO decision support problems

A SPO is characterized by its short duration (e.g. 5 years) and the constant need for answering 'what-if' questions in short time notice, for project redirection, and need for close project monitoring. Many of these activities of a SPO often involve interfacing with the various contractors across different functional divisions. With the exception of computerized engineering models, most of these activities and interfaces are currently performed manually. In this section, we examine sample decision problems that can arise within a SPO, as a first step in understanding the nature of a Decision Support System for aiding SPO decision-making.

(1) The budget for the current year is reduced by $ X, a SPO has to replan the project for this year and to obtain a new schedule. The effect of the budget reduction on the overall project plan has to be assessed. This entails detailed examination and evaluating tradeoffs to accomodate the budget changes across many different tasks.

(2) A SPO submits a budget for the current year, it has to supply details of current project status, as well as projected project status for the comming year. This includes milestones accomplished with the funds used up to date (Zero Base Budgeting), how and when has the funds been used, etc.

(3) A contractor proposes to change the engineering design (e.g. to use a new technology). A SPO has to evaluate the effect of this engineering change on the product quality, any additional costs it may incur, and any additional time required to complete the task.

(4) Part of the project (e.g. a work package or an account) overruns in cost/schedule. A SPO has to determine the impact on the overall schedule. The SPO may want to determine how much additional resources are required to bring the project on target again, or how to replan certain parts of the project to ease the impact of the over-run.

(5) The funding agency contemplates a budget cut for the coming year, a SPO is asked to assess the impact of this budget cut on the system development effort. Furthemore, the SPO is asked to assess the quality of the development effort if the planned project deadline is still to be met.

(6) The funding agency decides to reduce the scope of the project (e.g. build only a few prototype planes). A SPO is asked to resubmit a program plan and its associated cost estimates. Furthermore, the funding agency may like to know, given the current status of the project, the various project scopes possible, and their associated cost estimates (to determine the best course of action).

IV Resources for a SPO Decision Support System

Two basic resources for supporting critical SPO decision problems are the models and data. In this secion, we discuss the basic resources that are currently available to a SPO in dealing with the decision problems that a SPO often encounters. Then we examine additional basic resources for decision-making that may be used by a SPO to deal with these problems more effectively. In a latter section, we shall examine one approach of integrating some of these models and data in a SPO Decision Support System.

IV.1 Existing resources available to a SPO

A SPO often has considerable experience in developing, acquiring, and using computerized simulation models for evaluating performances of engineering designs. A wide variety of engineering simulation models are accessible to a SPO (possibly from previous projects or from other parts of the Air Force organization). However, the information systems used for planning and control of the project and for aiding critical decisions are typically manual.

Most large contractors have computerized information systems for monitoring their subproject progress, information about these subprojects of the overall SPO project is available to a SPO through contractor reports, and indirectly through SPO/contractor interface personnel. Hence the information flow from contractors

to a SPO is manual. This information constitutes the master files maintained by SPO Data Control, as discussed in an earlier section.

For project quality control, a SPO maintains a set of master files on all current engineering designs and records of all engineering change proposals and engineering change orders.

Historic information about SPO projects are kept by the Air Force organization and are available to a SPO for analysis. Fincancial accounting information and personnel information are also maintained by the Air Force organization and are available to a SPO.

Thus we see that there is a wide variety of engineering models and project management data bases in various forms available to a SPO. If these inforamtion and models can be made easily accessible and used jointly with the appropriate project process models, they can facilitate more effective decision-making on critical areas of SPO management.

IV.2 Potential resources available to a SPO

Many types of project process models may be used by a SPO for top management decision-making as well as within each functional divisions. For example, SPO top management may benefit from using planning models such as PERT, cost forecasting models, and

various resource tradeoff evaluation models.

The existing databases maintained by a SPO may be computerized so that information about project status can be obtained quickly for analysis. An automatic system for SPO Quality Control will make updating and coordinated evaluation of ECPs more effective.

The manual interface between contractor reporting and SPO project planning and control may be computerized to provide more effective exchange of information between a SPO and its contractors. For example, a SPO and its contractors may benefit from sharing certain common data, such as project status information.

With the relevant databases accessible in computerized form, a system that ties these databases and the various models together can be used effectively for aiding critical decisions of a SPO. We shall examine one approach to realize such a system.

V Framework for a Composite SPO Decision Support System

We have seen in the previous section that the databases and models that may be used to aid decision problems in a SPO may come from a variety of sources. A PERT system may be acquired from another part of the Air Force, a cost forecasting model may be developed by a SPO contractor, and the various databases discussed above come from many different systems. It seems desirable to have the capability of integrating these existing databases and models quickly and at low development cost, into a system that can be used to aid critical decisions that a SPO often has to make. We call such a system a Composite SPO Decision Support System (DSS). The reason for making use of existing databases and models for supporting decisions in a SPO is due to the nature of a SPO organization, its relatively short duration, and the nature of its decision problems. These characteristics have been discussed in previous sections.

Figure V.1 illustrates a Composite SPO Decision Support System. Three models of the system are shown: (1) a PERT system for evaluating project schedule, costs and resource usage, (2) a Tradeoff Model for assessing tradeoffs between manpower and product quality, and (3) a Cost Forecasting Model for forecasting costs of various tasks under various conditions. These models may have been acquired from very different sources, hence each may originally be operating under different operating environments and use different databases. In Figure V.1, four of

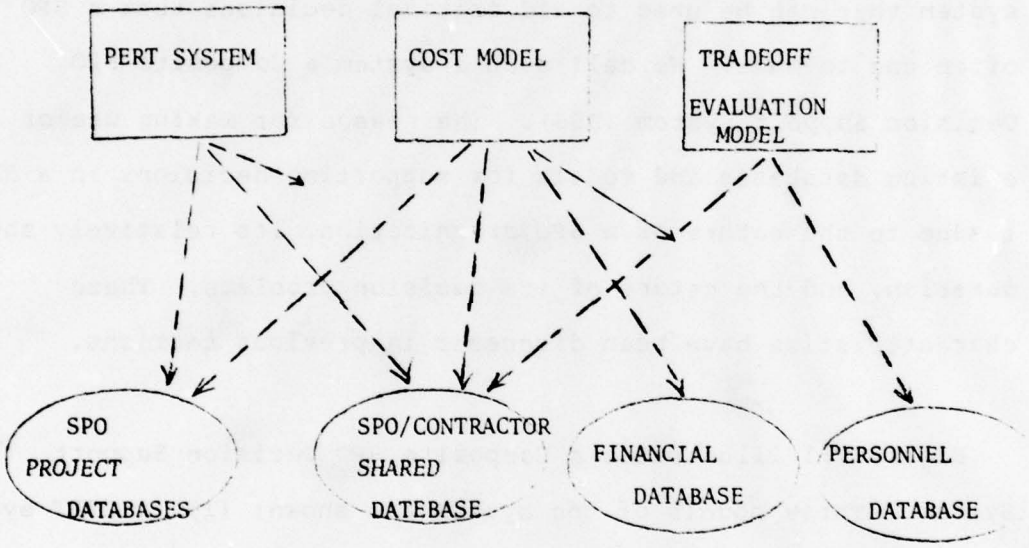


Figure V.1 Illustration of a composite SPO
Decision Support System

the databases in the Composite SPO DSS are shown. These are: the financial and personnel databases from the SPO supporting organization, the SPO's own project status database, and the database from the SPO/contractor interface.

Let us illustrate how this Composite SPO DSS may be used to meet a particular crisis. For example, the funding agency contemplates a budget cut of $ X. A SPO would like to assess the impact of this budget cut on the critical aspects of the project plan, such as : how to rearrange task priorities and how to tradeoff various resources with product quality to be still on schedule. Several possible scenarios of using this Composite SPO DSS follows:

(1) Non-critical tasks on the project plan may be deferred. Current cost changes associated with this action and the expected costs to be incurred by the task at a later date can be assessed using the Cost Forecasting Model. Thus the overall cost effects of this action can be determined.

(2) The various financial and personnel data may be obtained from the various databases to be used as input to the Tradeoff Model to determine the effects of the budget reduction on the quality of the product to be delivered.

(3) The effects on the overall schedule of several 'seemingly harmless' actions to be taken in response to the budget cut may

be examined using the PERT system. Unexpected bottlenecks on the critical tasks can be detected in advance.

VI Conclusions

From this preliminary investigation, it seems that a Composite SPO Decision Support System promises great potential in aiding a SPO to effectively make ad-hoc decisions. Subsequent studies will provide more detailed information on materials presented here.

Addendum Technical Report 5

COMPDATA : A Generalized High-Level Data Language System - Preliminary Investigations

Chat-Yu Lam

Stuart E. Madnick

I Introduction

For ad-hoc management decision support (Donovan and Madnick 77), it is often necessary to tap data from several different database systems. This need arises due to the nature of ad-hoc decision support and the fact that an organization often stores and processes data in separate databases that may be of different types and even geographically distributed. One key characteristic of ad-hoc decision support systems is the requirement for responsiveness to specific unforeseen problems, and their constantly changing decision support focus. An effective approach to meet these demand characteristics is to make jointly usable existing software systems and database systems (Donovan 76). Since existing software systems may not be compatible with one another, there is often a serious interfacing problem across system and data boundaries. Concepts for studying information systems that integrate existing, often incompatible software systems have been developed (Lam and Madnick 78).

The Generalized Management Information System (GMIS) (Donovan and Jacoby 75) (Lam and Madnick 78), for example, makes use of virtual machine technology to bring together existing database systems and models to aid energy policy decision-making in New England. GMIS hosts a variety of database systems each operating on a different virtual machine, for example, SEQUEL (Chamberlin and Boyce 74) and the CMS (IBM 76) file system. Figure I.1 illustrates the structure of GMIS. A user performing analysis in

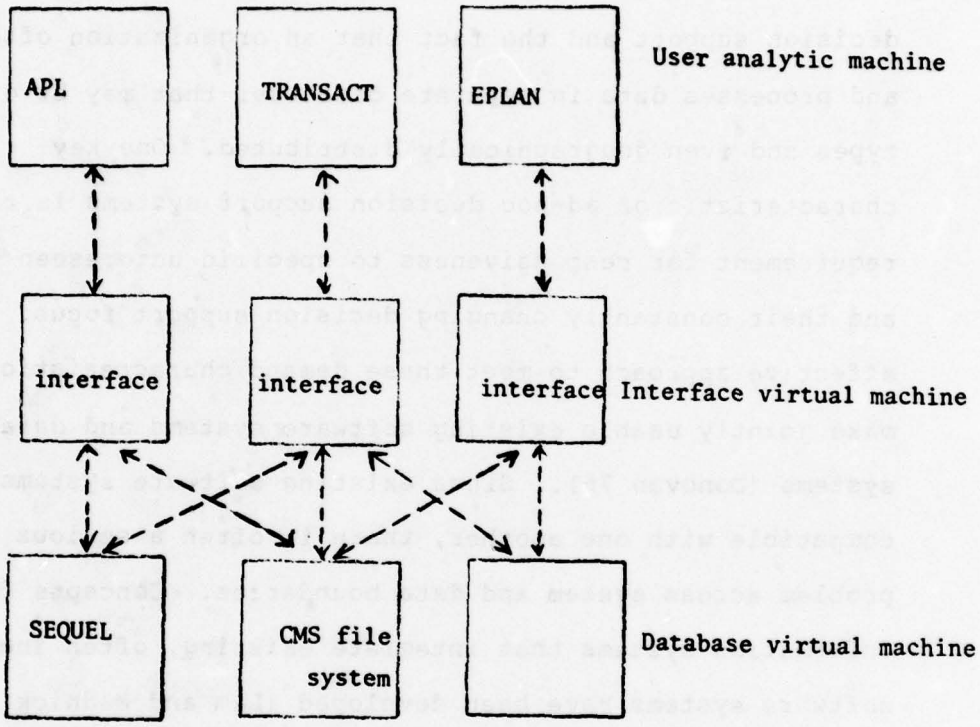


Figure I.1 Structure of GMIS

a virtual machine can access data stored in any one of GMIS's database systems, each operating on a different virtual machine. GMIS has been found to be very effective in supporting analysis of unanticipated decisions in the energy area.

Two major improvements to GMIS are desirable. First, there is a need to improve the user interface to GMIS. Currently, a GMIS user accessing different databases has to learn the different data languages and data structures that GMIS supports. It would be desirable to provide an unified means of accessing data that may be stored across different systems. Second, automating the development of interface virtual machines would allow rapid incorporation of new databases, and facilitate development of GMIS-type systems. The key purpose of an interface virtual machine is to resolve the data incompatibilities between a database virtual machine and a user virtual machine. The nature of these interface virtual machines are discussed in detail in a previous Technical Report (Technical Report No. 2).

As a first step in meeting these major improvements to GMIS-type systems, we introduce the Composite Database System, COMPDATA, currently being studied, that provides an unified means of accessing data. A user interacts with COMPDATA using a very high level language based on concepts that the user is familiar with. COMPDATA makes use of semantic information about the user's application domain and semantic information about the databases to generate the appropriate data operations on the

various databases (possibly stored in different systems) in satisfying the user's data requests.

In section II, components of COMPDATA are discussed. Section III illustrates use of COMPDATA by several examples. Section IV discusses the application of COMPDATA to GMIS-type systems as well as other database system configurations. Finally, we conclude with suggestions for further research.

II Structure of COMPDATA

The structure of COMPDATA is illustrated in Figure II.1. The COMPDATA language system consists of three major components: (1) semantic data system, (2) user language interface, and (3) database system(s) interface(s).

The semantic data system maintains and processes two types of semantic information: (1) user application domain semantics, and (2) database semantics. Its purpose is to support the language interface in understanding a user's data request (which is based on the user's view of information) and in mapping the user's data request to the appropriate data operations on stored data, possibly on different database systems.

The user language interface provides a high-level data language based on concepts in the user's application domain. This interface also determines the appropriate data operations on the databases to satisfy a user data request. The user language interface views all stored data as virtual relations. Therefore a user data request is translated into operations on these virtual relations.

The database system interface in COMPDATA maps data operations on virtual relations into the actual data manipulation operations on real data. Each of these components of COMPDATA are discussed in the following subsections.

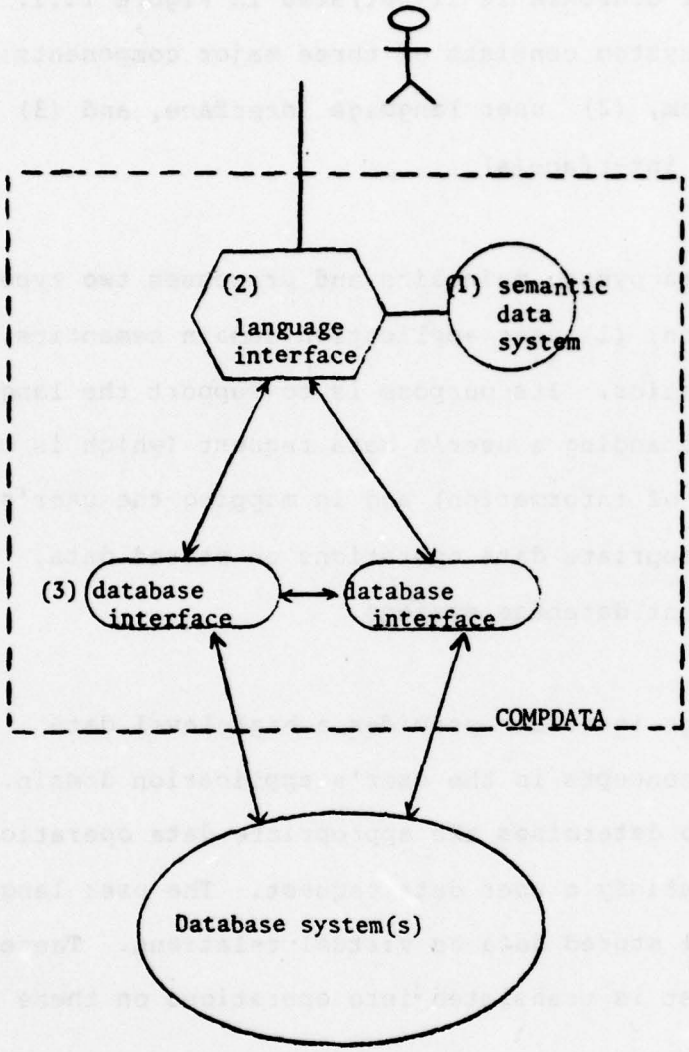


Figure II.1 Structure of COMPDATA

II.1 Semantic data system

The semantic data system maintains semantic information about the user's application domain and semantic information about the databases. Semantic information about the user application domain describes entities and associations among entities. These are the knowledge that the user is assumed to possess when requesting data from COMPDATA. Semantic information about databases describes the actual represenatation of the stored data. Both types of semantic information are used by the COMPDATA language interface to determine the appropriate data manipulation operations corresponding to a user's data request.

(1) User's application domain semantics

For the purpose of this report, we shall illustrate a plausible representation scheme for the user's application domain semantics. This representation scheme is based on the Entity-Relationship (E-R) model (Chen 76), with extensions by (Lee, 78), and some of our extensions to the E-R representation. These ideas are briefly discussed below.

Using the E-R model, the user application domain consists of entities. An entity has attributes and entities may be related to one another. A relationship among entities may also have attributes. Figure II.2 illustrates an E-R representation of a simple application domain consisting of two entities, car and

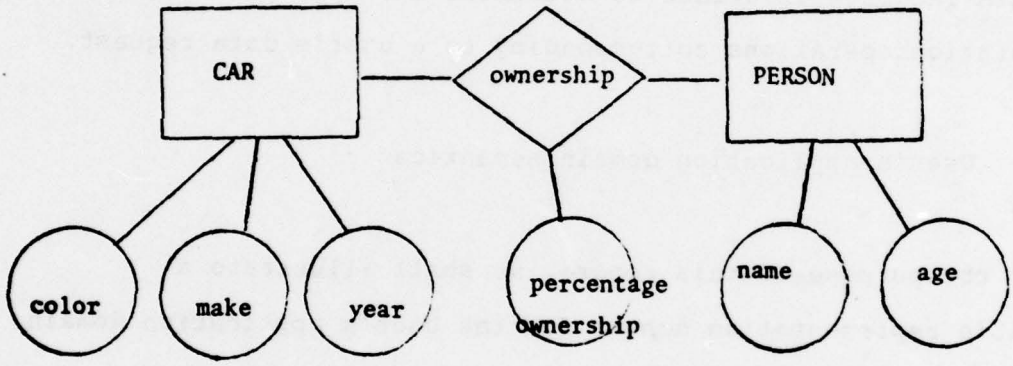


Figure II.2 An E-R represenation of user application domain semantics

person. These entities are related to each other, namely, a person may own a car and a car may be owned by a person. The attributes of these entities are self-explanatory.

Lee extended the E-R representation to capture the subsetting properties among entities. For example, an entity may be partitioned into classes of entities according to values of its attributes. Figure II.3 illustrates two possible partitions of the person entity. The entity person can be partitioned into two entities, student and faculty according to type of person. The entity person can also be partitioned into several other entities according to the department in which a person belongs.

Another type of semantic information that is not captured by the above representation scheme is often refered to as semantic integrity constraints. Lee proposed to use predicate calculus on the entities and relationships to represent these additional semantics. For example, for a particular user's application domain, it may be important to know that a person's salary cannot exceed certain limits. A possible representation of this semantic constraint is as follows:

∀ PERSON i : (i.salary LESS-THAN 50000)

Finally, we propose another extension to the E-R representation to allow for the definition of new entities from existing entities and relationships. For example, a new entity, delinquent-account, may be defined as:

ACCOUNT x : (x.balance GREATER-THAN 500) AND
(x.due-date EQUAL-TO 1976)

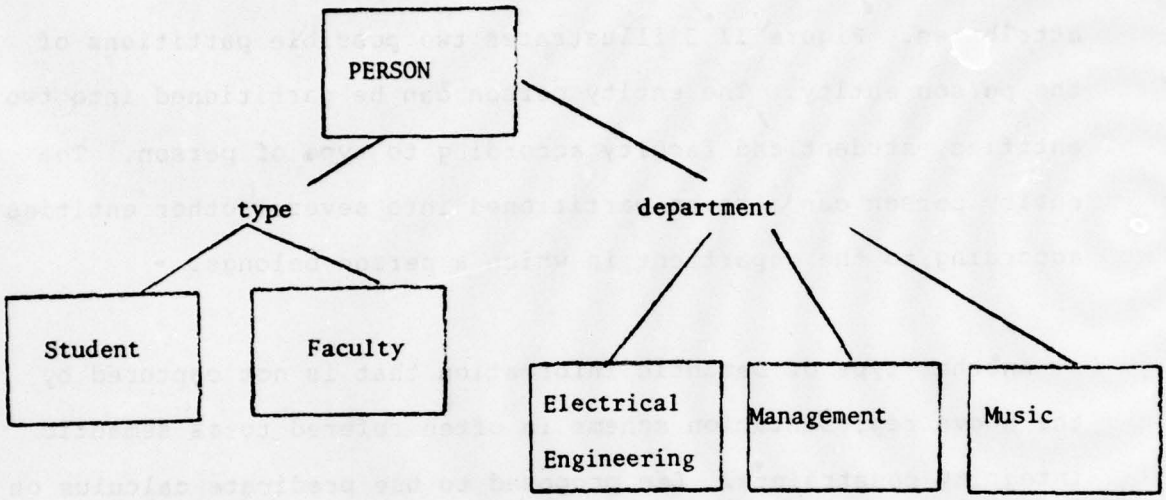


Figure II.3 Partitioning of an entity into a hierarchy of entities

We have illustrated a plausible scheme for representing user's application domain semantics. Next, we examine the problem of representing data semantics.

(2) Database semantics

As indicated in Figure II.1, COMPDATA interacts with the user on one hand, and interacts with database systems on the other hand. To effectively realize a user data request, COMPDATA has to maintain a data dictionary of the various databases. This task is further complicated when the databases are stored in multiple database systems since the same entity may be represented differently in different database systems. For example, the entity person may be modelled as one record type in a particular database system, while it may be modelled as two different record types, *student* and *faculty*, in another database system. The exact meanings of existing data fields, domains, etc, must be explicitly represented to support operations in COMPDATA. For example, let us consider the meaning of the following relation:

RX (name-x, name-y, salary-1, salary-2, year)

Without explicit representation of the data semantics, domains in RX could have been taken to mean different things by different users. For instance, name-x may be taken to be the name of a department in a company where the person with name-y works.

Salary-1 and salary-2 may be taken to be the previous salary and present salary of the person respectively. Year may be taken to be the year the person joined the company, or to be the date of birth of the person, or ....

Figure II.4 illustrates an explicit representation of the relation RX, using the E-R notation. The relation RX actually represents a marriage relationship between two persons. Name-x is the name of the husband, name-y is the name of the wife, salary-1 is the salary of the husband, salary-2 is the salary of the wife, and year is the year of their marriage. By maintaining information on the semantics of stored data, COMPDATA is able to relieve the user of the unpleasant, sometimes impossible task of keeping track of all the meanings of data in the various database systems.

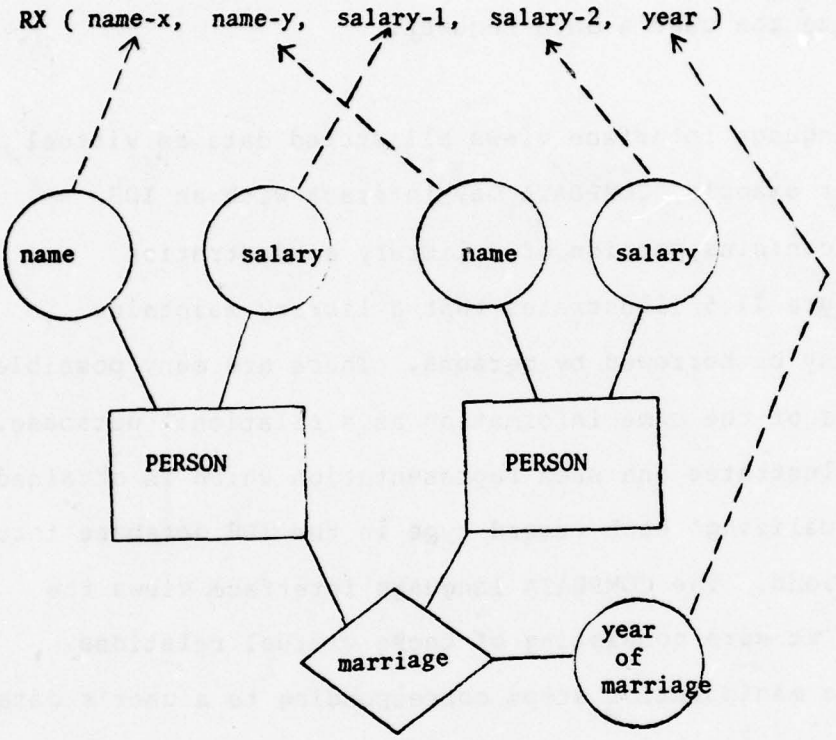


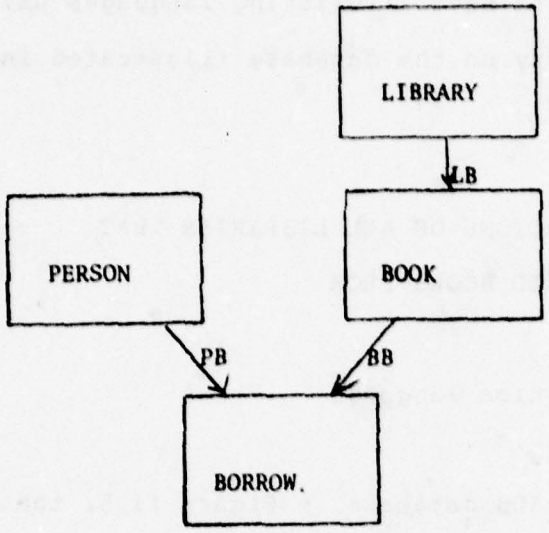
Figure II.4 Data Semantics of relation RX

II.2 User data language interface

The COMPDATA user data language interface performs two basic functions: (1) interacts with the user to obtain the user's data requests, and (2) determines the data manipulation operation steps to realize the user's data request.

The data language interface views all stored data as virtual relations. For example, COMPDATA may interact with an IDS database that contains portion of a library adminstration database. Figure II.5 illustrates that a library maintains books, which may be borrowed by persons. There are many possible representations of the same information as a relational database. Figure II.5 illustrates one such representation which is obtained by simply 'normalizing' each record type in the IDS database into separate relations. The COMPDATA language interface views the IDS data as if it were consisting of these virtual relations. Hence, the data manipulation steps corresponding to a user's data request operate on these virtual relations. There is another component of COMPDATA that maps these operations on virtual relations to actual operations on the real data. For example, a relational JOIN operation may be mapped into a series of IDS data manipulation steps, such as FIND UNIQUE, FIND OWNER, etc. This component of COMPDATA will be discussed in a later section.

As a first step in developing a generalized language for COMPDATA, we examine several existing languages and then



(a) IDS Network Data Model

PERSON (name, type, address, ...)

LIBRARY (name, location, ...)

BOOK (call-no, title, author, ...)

BORROW (call-no, name, due-date, borrow-date, ...)

(b) COMPDATA Relational Data Model

Figure II.5 Partial IDS library database
and COMPDATA's relational view of the data

illustrate the nature of a plausible language for COMPDATA.

We shall illustrate several existing languages using the following sample query on the database illustrated in Figure II.5:

```
' FIND THE LOCATIONS OF ALL LIBRARIES THAT
  'CHAT' BORROWED BOOKS FROM
```

(1) Record-at-a-time language

Referring to the IDS database in Figure II.5, the following IDS-like data language steps will satisfy the sample query.

```
1.  FIND PERSON = 'CHAT'
2.  FIND NEXT BORROW OF SET PB, IF EOF STOP
3.  FIND OWNER OF SET BB
4.  FIND OWNER OF SET LB, PRINT LOCATION
5.  GOTO 2
```

(2) Relational algebra language

Refer to the virtual relations illustrated in Figure II.5 that correspond to the partial library database, the following relational algebra steps will accomplish the sample query.

```
1.  T1 = JOIN BORROW (NAME = 'CHAT')
2.  T2 = JOIN T1 (CALL_NO ) BOOK
3.  T3 = PROJECT T2 ON (CALL_NO , NAME)
4.  T4 = JOIN T3 (NAME) LIBRARY
5.  T5 = PROJECT T4 ON (CALL_NO, LOCATION)
6.  PRINT LOCATION IN T5
```

(3) Relational calculus

Using the virtual relations in Figure II.5, the following SEQUEL (Chamberlin and Boyce 74) relational calculus specification will satisfy the sample query.

```
SELECT LOCATION FROM LIBRARY WHERE NAME =
  SELECT NAME FROM BOOK WHERE CALL_NO =
    SELECT CALL_NO FROM BORROW WHERE NAME = 'CHAT'
```

(4) Languages based on the E-R model (Chen 76)

(a) A Conceptual Language for Entities and Relationships (CLEAR) (Poonen 78)

The library database in Figure II.5 is represented as an E-R database in Figure II.6. The following CLEAR query can be used to satisfy the sample query.

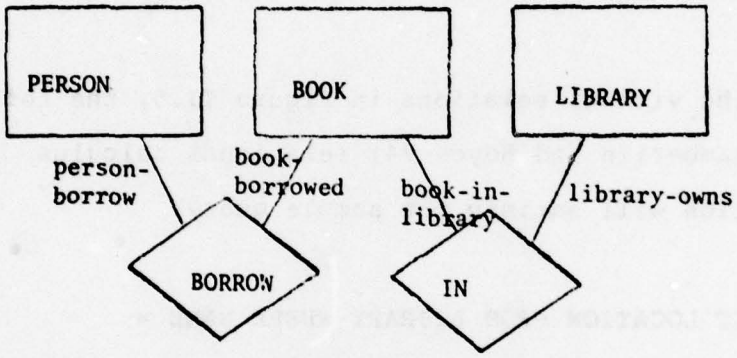


Figure II.6 An E-R representation of the library administration database

```
LOCATION OF LIBRARY-OWNS WITH CALL_NO OF BOOK-IN-LIBRARY =
CALL_NO OF BOOK-BORROWED WITH NAME OF PERSON-BORROW = 'CHAT'
```

(b) A Chain Based Language (CABLE) (Shoshani 78)

Using the same E-R database as above the following CABLE query will satisfy the sample query.

```
OUTPUT LOCATION
SELECT LIBRARY/BOOK/PERSON='CHAT'
```

(5) Natural English

There has been considerable research interest in using natural English as a data language. A database system that carries on English dialogs with the user is being investigated (Codd 74). With a true natural English query system, it would be possible to directly satisfy the original sample query :

```
FIND THE LOCATIONS OF ALL LIBRARIES THAT
'CHAT' BORROWED BOOKS FROM
```

This cursory exposition of the above data languages is by no means a fair comparison of these languages. In summary, the

record-at-a-time type of languages require that the user has fairly detailed knowledge of the access paths to obtain the data. The relational model does not represent relationships among tables explicitly, thus the user is required to specify these relationships (via common domains) when the query is specified. The E-R model represents relationships explicitly, thus sometimes the user may not be required to specify these relationships explicitly in the query formulation, however the user must be aware of these relationships and what they mean.

A language for COMPDATA is being developed. In this report, we shall illustrate the nature of such a language. The COMPDATA language is based on the principle that a user formulates a data request using only one entity or one relationship among several entities. For requesting data that may involve more than one entity or relationship, the user will issue several successive data requests, using temporary results of each data request in the other data requests. (In terms of relational databases, this would mean that the user will not issue a data request that involves more than a few relations at a time) Thus, a user's request will be formulated as operations on a single entity, or as operations on a relationship and its associated entities.

We shall illustrate the flavor of the COMPDATA language with several examples. The following notations regarding the syntax of data requests will be used: keywords are underlined, simple values are in quotes, and entity names are capitalized. For the

examples shown in this report, we use the following syntax for a data request:

Select attributes of (all/any) ENTITY with
qualifying expressions

A qualifying expression is used to qualify an entity. An entity may be qualified by specifying values of its attributes. An entity may also be qualified by describing its relationship with other entities.

Example (1) : Find the names and ages of all persons.

Select name, age of all PERSON

Name and age are attributes of the entity person. There is no further qualification of person in this example.

Example (2) : Find the names of all male persons, 24 years of age.

Select name of all PERSON with age='24', sex='male'

The with clause in this example is used to further qualify person, namely, those with attribute age having a value of '24' and attribute sex with a value of 'male'.

Example (3) : Find the name of the oldest person.

Select name of PERSON with age greater-than
age of all PERSON

In this example, the value for age is a compound expression. Each person's age is compared with all other person's age until the oldest person is found.

Example (4) : Find the names of persons who are younger than their wives.

Select name of all PERSON with age less-than
age of PERSON's wife

The value for age is a compound expression evaluated via a relationship. This is the marriage relationship between two persons. Note the use of PERSON's wife to describe the marriage relationship and the role of the second person as a wife to the first person (the person being selected).

Example (5) : Find any employee who makes more than all the employee's manager(s).

Select name of any EMPLOYEE with salary greater-than
salary of all EMPLOYEE's manager

The value for the salary attribute is a compound expression being evaluated via a manager-worker relationship among employees.

Example (6) : Find the title of the book borrowed by 'Chat' from a library in 'Arlington', the due-date of the book borrowed is 'June 24,1978'.

```
Select title of BOOK borrowed-by PERSON with name='Chat'
  from LIBRARY with location='Arlington' where
    due-date='June 24,1978'
```

In this example, the entity book is explicitly qualified by a description of its relationship with two other entities, person and library. The where clause further qualifies this relationship. Note that due-date is an attribute of the relationship and not of any of the entities participating in the relationship.

Example (7) : Find the due-date of the book called 'Operating Systems' borrowed by 'Chat' from the library in Arlington on June 6, 1978.

```
Select due-date of BOOK with title ='Operating Systems'
  borrowed-by PERSON with name='Chat' from LIBRARY with
    location='Arlington' where borrow-date='June 6,1978'
```

The due-date in this example is an attribute of the relationship

among person, book and library. The syntax of this data request is the same as Example (6) where the attribute title is for the entity book only. This will not cause any ambiguity since there is no attribute of book that is also called due-date.

We have illustrated some data retrieval capabilities of the COMPDATA language. The capabilities to define new entities from existing entities and relationships have been briefly discussed in a previous section. Using these capabilities, it is expected that very complex data requests can be synthesized in a straight forward manner.

II.3 Database system(s) interface(s)

A database system interface exists in COMPDATA for each database system that COMPDATA interacts with. The function of this database interface is to realize the operations on virtual relations by mapping these operations into the appropriate data manipulation operations on the real data. This entails interacting with the database system as well as other database system interfaces. For example, suppose that COMPDATA interacts with two database systems (Figure II.7), and IMS database and an IDS database. The IDS database has been discussed before. The IMS database represents portion of an university's registrar database which contains information on departments, faculty, and students. A faculty may teach several courses and participate in several research projects. A student may be enrolled in several

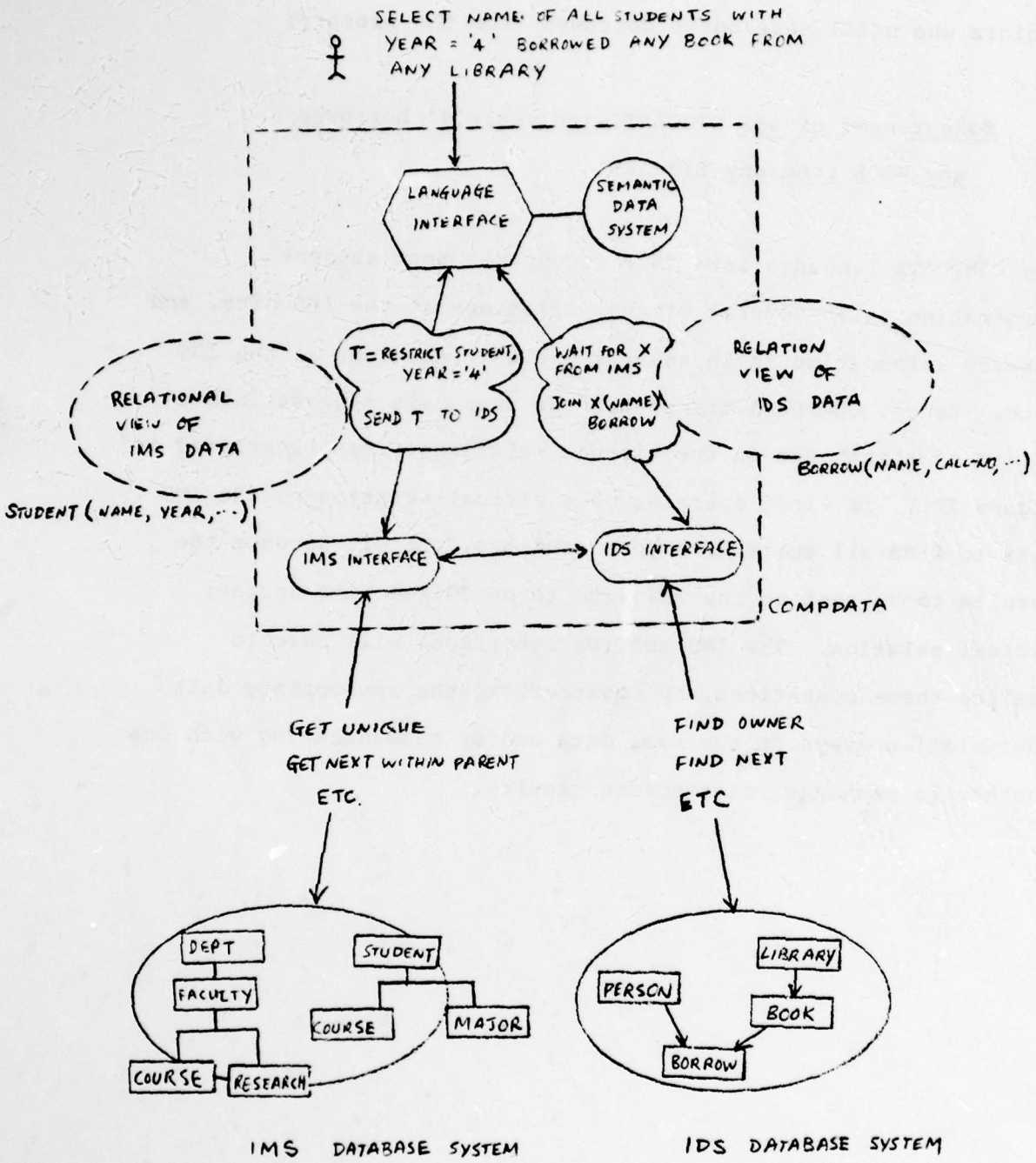


Figure 11.7 ILLUSTRATION OF COMPDATA OPERATION

courses and may have several majors. Suppose that the user issues the following data request to find all the names of the seniors who still have books borrowed from the library:

Select name of all STUDENT with year='4' borrowed
any BOOK from any LIBRARY

The COMPDATA language interface recognizes that student information is in several virtual relations at the IMS site, and library information is in several virtual relations at the IDS site. Hence, COMPDATA translates the user data request into a series of operations on the virtual relations. As illustrated in Figure II.7, it first operates on a virtual relation at the IMS site to find all the fourth year students, then it directs the results to be sent to the IDS site to be JOINed with another virtual relation. The IMS and IDS interfaces will have to realize these operations, by constructing the appropriate data manipulation steps on the real data and by communicating with one another to exchange intermediate results.

III Examples of using COMPDATA

In this section, we use several examples to illustrate the kinds of semantic information that may be required in translating a user request to its corresponding operations on virtual relations. We will use the following operators on virtual relations : (1) JOIN, (2) PROJECT, and (3) RESTRICT.

JOIN operates on two virtual relations, A and B, to produce a new virtual relation C, based on a domain 'd' common to A and B. A tuple of C is obtained by concaternating a tuple from A and a tuple from B that have the same value for the domain 'd'.

PROJECT operates on a single virtual relation to produce a subset of the virtual relation by eliminating some domains of the virtual relation.

RESTRICT operates on a virtual relation to form a new virtual relation which is a subset of the original virtual relation by selecting those tuples that satisfy certain criteria.

Example (1) : A user may specify the following request:

Select DELINQUENT-ACCOUNTS

Suppose the Semantic Data System contains the meaning of delinquent-accounts as accounts with balance-due greater than

'500' and due-date equal to '1976'. The COMPDATA database manager is then able to formulate the following operations on the appropriate virtual relation:

```
ANSWER = RESTRICT ACCOUNT : balance-due greater-than '500'
  and due-date = '1976'
```

Example (2) : A user specifies the following request:

```
Select name of STUDENT with age='24'
  enrolled-in COURSE with course-number ='15.565'
```

Suppose COMPDATA knows that there exists two virtual relations as follows:

```
COURSE (student, course)
STUDENT (name, age)
```

Suppose COMPDATA also knows that student in the COURSE relation and name in the STUDENT relation represent the same entity, then COMPDATA is able to form the following operations on the virtual relations:

```
TEMP1 = JOIN COURSE (student, name) STUDENT
TEMP2=RESTRICT TEMP1 : course='15.565' and age='24'
ANSWER = PROJECT TEMP2 (name)
```

Example (3) : Suppose that there are two virtual relations in COMPDATA as follows:

R1 (name, type, age)

R2 (student, course)

A user issues the same request as in the above example. COMPDATA knows that the R2 relation contains names of person and that person can be partitioned according to the type attribute into student and faculty, hence COMPDATA is able to specify the following operations on the virtual relations to accomplish the user request:

TEMP1 = RESTRICT R2 : type='student'

TEMP2 = JOIN TEMP2 (name, student) R1

ANSWER = RESTRICT TEMP2 : course='15.565' and age='24'

Example (4) : Suppose there are the following virtual relations in COMPDATA:

JAPAN (firm, asset)

USA (firm, asset)

A user specifies the following request, to obtain the total asset of the ACME coporation:

Select total (asset) of FIRM with name='ACME'

COMPDATA knows that the asset domains in the virtual relations JAPAN, and USA are of different units of measure, hence a conversion procedure has to be used before the asset values are summed. The following operations on the virtual relations may be used:

TEMP1 = RESTRICT JAPAN : firm='ACME'

TEMP2 = RESTRICT USA : firm='ACME'

(use a conversion procedure to convert asset in TEMP1 and TEMP2)

(use a procedure to sum the asset values in TEMP1 and TEMP2)

Example (5) : Suppose we have the following virtual relation in COMPDATA :

FUEL-STORAGE (city, tank-type, tank-volume)

A user then specifies the following request to find the capacity for no-2-oil in Boston:

Select total (tank-volume) of FUEL-STORAGE
with city='Boston', fuel-type='no-2-oil'

COMPDATA knows that no-2-oil can only be stored in tank-type

'X-100', hence COMPDATA is able to generate the following operations on virtual relations corresponding to the user's request:

```
TEMP1 = RESTRICT FUEL-STORAGE : city='Boston' and
  tank-type='X-100'
TEMP2 = PROJECT TEMP1 (tank-volume)
(use a procedure to sum the tank-volume values)
```

IV Applications of COMPDATA to different database system configurations

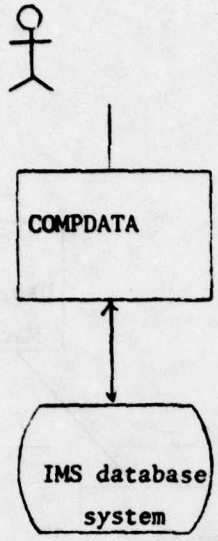
COMPDATA is a data language system that is applicable in many database system configuration. Three types of these configurations are discussed below.

(1) Single database system

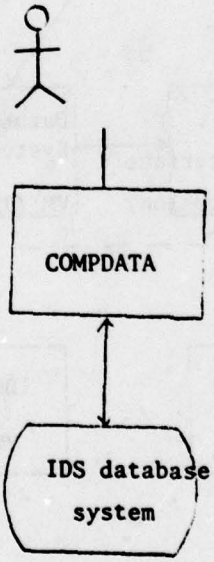
COMPDATA may be used as a high-level language interface to an existing database system. The user can use a consistent data language independent of the data schema that the database system supports, thus enhances logical data independence. For example, Figure IV.1 illustrates the use of COMPDATA as a language interface to an IMS database system and then to an IDS database system when the IMS database system is replaced by the IDS database system. The user is not aware of such changes because (s)he still accesses data using the same view of information that is most convenient for the application.

(2) Multiple database systems on a virtual machine system

A major motivation of developing COMPDATA is to improve the effectiveness of GMIS-type systems for decision support, in particular, to provide an easy to use interface for accessing data, and to improve the joint usage of multiple, different database systems. Figure IV.2 shows one possible scheme of using



Before using IDS database system



Changed from IMS to IDS database system

Figure IV.1 COMPDATA in single database system

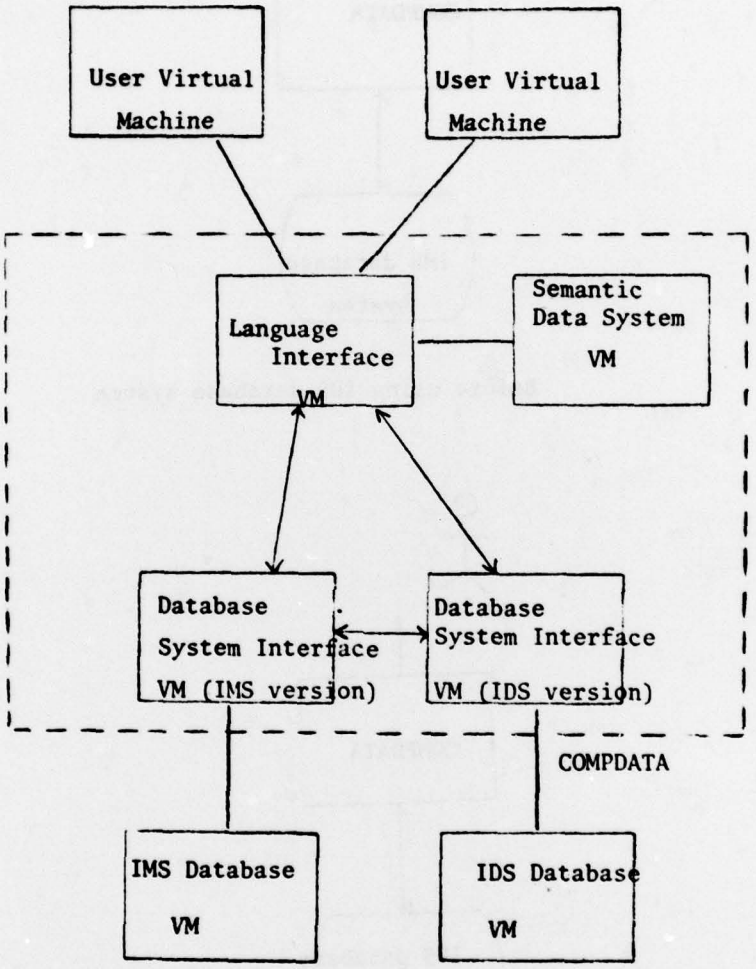


Figure IV.2 COMPDATA in GMIS-type system

COMPDATA in a GMIS-type system. The Language Interface Virtual Machine interacts with any User Analysis Virtual Machine to provide an unified high-level language interface to all stored data in the system. The Semantic Data System Virtual Machine maintains information on the user application and information on the various databases so that the language interface can interpret a user data request. A user data request is mapped into the appropriate operations on the virtual relations corresponding to the stored data. Each Database System Interface Virtual Machine is responsible for realizing these operations on virtual relations by interacting with the Database Virtual Machines. Using such a scheme, it appears to the user that (s)he is interacting with a single, easy to use database system.

(3) Distributed database systems

COMPDATA is applicable to distributed database system configurations. For distributed database systems without data redundancy, COMPDATA functions just like it does in the GMIS-type system configuration discussed above, except that the virtual machines are replaced by real machines. For distributed database systems with data redundancy, translation of a user data request to operations on the various database systems may be very complex. On the other hand, there is significant benefit to use COMPDATA in distributed heterogeneous database systems since it is unlikely that the user is willing or able to keep an up to date knowledge of the various data structures maintained by each

database system. Use of COMPDATA enables the user to obtain data using only knowledge about the application that (s)he is familiar with.

V Conclusions

We have introduced a system that provides an unified means for accessing data. This system is currently under study. This system makes use of semantic information about the user's application domain and the databases to translate a user's data request to the appropriate data operations on the databases. The user is not required to know where or how the data is actually stored.

Three major research areas have been identified associated with COMPDATA. These are : (1) specification of a data language for COMPDATA, (2) development of semantic representation scheme for COMPDATA, and (3) mapping of operations on virtual relations to operations on actual data which may be physically distributed on heterogeneous database systems. We are actively investigating (1) and (2).

REFERENCES

(Alter, 1975)
Alter, S.: "A study of computer-aided decision making in organizations," unpublished doctoral thesis, MIT Sloan School of management, 1975.

(Anthony, 1965)
Anthony, R. N.: "Planning and Control Systems: A Framework for Analysis," Harvard University Graduate School of Business Administration, Boston, 1965.

(Arthur Anderson & Co. 77)

Arthur Anderson & Co. "An Approach to Data Independence, Arthur Anderson & Co. internal report, February, 1977, Chicago, Ill.

(Astrahan et al, 1976)

Astrahan, M. M., et al: System R: Relational Approach to Database Management, ACM Trans. on Database Systems 1, 2, June, 1976.

(Buneman le. al. 77)
Buneman O.P., Morgan H.L., and Zisman, M.D. : Display Facilities for DSS Support: The DAISY Approach, DATABASE, Vol. 8, No. 3, 1977, 46-50.

(Cambridge, Project 74a)

The Cambridge Project : Handbook of Prograr and Data, Cambridge, Mass.

(Cambridge Project 74b)

The Cambridge Project : JANUS Technical Reference, Cambridge, Mass.

(Cambridge Project 74c)

Cambridge Project Annual Report, Cambridge, Mass, 1974.

(Chamberlin and Boyce 74)
Chamberlin, D.D., and Boyce, R.F. : SEQUEL: A Structured English Query Language, Proceedings ACM SIGFIDET Workshop, Ann Arbor, MI, May 1974, 249-264.

(Chen 76)

Chen, P.S. : The Entity-Relationship Model - Toward a Unified View of Data, ACM Transactions on Database Systems, Vol. 1 No. 1, March 1976, 9-36.

(CODASYL 71)

CODASYL Data Base Task Group, April 1971 report, ACM, New York City, 1971.

(Codd 75)

Codd, E.F. : Seven Steps to Rendezvous with The User, Proc. IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, April 1974, North-Holland.

(Crick et al 70)

Crick, M.F.C., Lorie, R.A., Mosher, E.J., Symonds, A.J.: A database system for interactive applications, IBM Cambridge Scientific Center, Report No. G320-2058, July 1978.

(Crowston 71)

Crowston, W.B. : Models for Project Management, Sloan Management Review, Spring 1971, Vol. 12, No. 3.

(Donovan, 1972)
Donovan, J. J.: Systems Programming, McGraw-Hill, New York, 1972.

(Donovan 76a)
Donovan J.J. : Database System Approach to Management Decision Support, ACM Transactions on Database Systems, Vol 1 No. 4, December 1976, 344-369.

(Donovan 76b)

NEEMIS Text of Governors Presentation of October 6, 1975, MIT ENergy Laboratory Working Paper No. MIT-EL-76-00WP, Cambridge, Mass., February, 1976.

(Donovan 76c)

Donovan, J.J.: A note on performance of VM/370 in the integration of models and databases, MIT Sloan School of Management, CISR Report No. 26, June 1976.

(Donovan and Madnick 76a)

Donovan, J.J. and Madnick, S.E. : Institutional and Ad-hoc Decision Support Systems and Their Effective Use, DATABASE, Vol. 8, No. 3, Winter 77, 79-88.

(Donovan and Jacoby, 1975)
Donovan, J.J. and Jacoby, H.D.: "Virtual Machine Communication for the Implementation of Decision Support Systems," MIT Sloan School of Management, CISR Report No. 28, Dec., 1976.

(Donovan and Madnick 76a)

Donovan J.J., and Madnick, S.E. : Virtual Machine Advantages in Security, Integrity, and Decision Support Systems, IBM Systems Journal, Vol 15 No. 3, 1976, 270-278.

(Donovan and Madnick 77)

Donovan, J.J. and Madnick, S.E. : Institutional and Ad-hoc Decision Support Systems and Their Effective Use, DATABASE,Vol. 8, No. 3, Winter 77, 79-88.

(ECM 78)

The ECM building energy conservation analysis computer program: User's Guide, MIT Energy Laboratory, Cambridge, Mass., January Þ 78.

(Forrester 75)

Forrester, J.W.: Dynamics of Socio-economic Systems, Report No. D-2230-1, MIT System Group, Cambridge, Mass, August, 1975.

(Gerrity 71)

Gerrity, T.P.: Design of Man-machine Decision Systems: An Application to Portfolio Management, Sloan Management Review, Winter, 1971, Þ 75.

(Goldberg, 1974)

Goldberg, R. P.: Survey of Virtual Machine Research, Computer, June, 1974.

(Gorry and Scott Morton, 1971)

Gorry, G.A. and Scott Morton, M.S.: A framework for management information systems, Sloan Management Review, Fall, 1971.

(Gray and Watson, 1975)

Gray, J.N. and Watson, V.: "A shared segment and inter-process communication facility for VM/370," Res. Rep. RJ 1579, IBM Res. Lab, San Jose, Calif., Feb., 1975.

(Guido and Considine 77)

Guido, A.A. and Considine, J.: Laboratory automation via a VM/370 teleprocessing virtual machine, National Computer Conference, Þ 77, 865-877.

(Hsieh, 1974)

Hsieh, S.C.: "Inter-virtual machine communication under VM/370," Form No. RC 5147, IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., Nov., 1974.

(IBM, 1972)

IBM: IBM virtual machine facility/370: Introduction, Form No. GC20-1800, IBM, White Plains, N.Y., 1972.

(IBM, 1976)

IBM: IBM virtual machine facility/370: CMS user's guide, Form No. GC20-1819, IBM White Plains, N.Y., 1976.

(IBM 1977)

IBM virtual machine facility/370: System programer's guide, Form No. GC20-1807, IBM, White Plains, N.Y., 1977.

(Lam and Madnick, 1978a)

Lam, C.Y. and Madnick, S.E.: "Use of Virtual Machines for Development of Decision Support Systems: Strategies for Interfacing Virtual Machines," CISR internal report NO. R001-7804-01, MIT Sloan School of Management, 1978.

(Lam and Madnick, 1978b)

Lam, C.Y. and Madnick, S.E.: "Strategies for Interfacing Virtual Machines - A Case Study," CISR internal report No. R001-7804-02, MIT Sloan School of Management, 1978.

(Lam and Madnick, 1978c)

Lam, C.Y. and Madnick, S.E.: "Composite Information Systems - A New Concept in Information Systems," CISR Report No. 35, MIT Sloan School of Management, 1978.

(Lam and Madnick, 1978d)

Lam, C.Y. and Madnick, S.E.: "Preliminary Investigations of a SPO Decision Support System," CISR internal report No. R001-7807-04, MIT Sloan School of Management, 1978.

(Lam and Madnick, 1978e)

Lam, C.Y. and Madnick , S.E.: "COMPDATA: A Generalized High-Level Data Language System - Preliminary Investigation," CISR internal report No. R001-7807-05, MIT Sloan School of Management, 1978.

(Lee 78)

Lee, R.M. and Gerritsen, R.: A Hybrid Representation for Database Semantics, Working Paper No. 78-10-10, Department of Decision Sciences, Wharton School, University of Pennsylvania, Philadelphia, Penn. 19104.

(Levitt et.al. 74)

Levitt, G., Stewart, D.H., Yorkmark, B.: A Prototype for Interactive Data Analysis, National Computer Conference, 1974, 63-69.

(Lorie 74)

Lorie, R.A.: XRM - An extended (N-ary) relational memory, IBM Cambridge Scientific Center, Report No. 320-2096, January, 1974.

(MacAvoy and Pindyck 75)

MacAvoy, P.W., and Pindyck, R.S.: Price Controls and The National Gas Shortage, American Enterprice Institute for Public Policy Research, Washington, D.C., 1975.

(Madnick, 1970)

Madnick, S.E.: "Design Strategies for File Systems," MIT Project MAC TR-78, Oct., 1970.

(Madnick, 1974)

Madnick, S.E.: Operating Systems, McGraw-Hill, New York, 1974.

(MCA 76)

Massachusetts Computer Associates, Inc.: Second Semi-annual Report, August, 1976, Wakefield, Mass.

(NBER 74)

National Bureau of Economic Research, Inc.: ACOS Overview, 1974, Cambridge, Mass.

(NBER 75a)

National Bureau of Economic Research, Inc.: ACOL Reference Manual, 1975, Cambridge, Mass.

(NBER 75b)

National Bureau of Economic Research, Inc.: TROLL Primer, 1975, Cambridge, Mass.

(NBER 75c)

National Bureau of Economic Research, Inc.: DASEL Users Guide, 1975, Cambridge, Mass.

(Poonen 78)

Poonen, G.: CLEAR: A Conceptual Language for Entities and Relationships, Digital Equipment Corporation, Maynard, Mass., 1978.

(Roos 67)

Roos, D.: ICEC System Design, The MIT Press, 1967, Cmabridge, Mass.

(Schoderbek and Digman 67)

Schoderbek, P.P. and Digman, L.A.: Third generation, PERT/LOB, Harvard Business Review, September-October 1967, 100-110.

(Scott Morton, 1971)

Scott Morton, M.S.: "Management Decision Systems: Computer Based Support for Decision Making," Div. of Res., Grad. Sch. of Bus. Adm., Harvard U., 1971.

(Shoshani 78)

Shoshani, A.: CABLE: A Language Based On The Entity-Relations Model, personal working paper, 1978.

(Simon, 1960)

Simon, H.A.: The New Science of Management Decision, Harper & Row, New York, 1960.

☆U.S. GOVERNMENT PRINTING OFFICE:1979-614-033/330

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.